

RL-TR-91-397
Final Technical Report
December 1991

DTIC
ELECTED
JUL 8 1992
AD-A252 777



2

INTEGRATION OF TOOLS FOR THE DESIGN AND ASSESSMENT OF HIGH- PERFORMANCE, HIGHLY RELIABLE COMPUTING SYSTEMS (DAHPHRS)

Research Triangle Institute

Sponsored by
Strategic Defense Initiative Office

92-17698



APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

92 7 07 002

The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Strategic Defense Initiative Office or the U.S. Government.

Rome Laboratory
Air Force Systems Command
Griffiss Air Force Base, NY 13441-5700

This report has been reviewed by the Rome Laboratory Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RL-TR-91-397 has been reviewed and is approved for publication.

APPROVED:

Patrick O'Neill

PATRICK J. O'NEILL
Project Engineer

FOR THE COMMANDER:

John A. Graniero

JOHN A. GRANIERO
Chief Scientist
Command, Control, & Communications Directorate

If your address has changed or if you wish to be removed from the Rome Laboratory mailing list, or if the addressee is no longer employed by your organization, please notify RL (C3AA) Griffiss AFB NY 13441-5700. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

INTEGRATION OF TOOLS FOR THE DESIGN AND ASSESSMENT OF HIGH-PERFORMANCE, HIGHLY RELIABLE COMPUTING SYSTEMS (DAHPHRS)

**Charlotte O. Scheper
Robert L. Baker
Harold L. Waters II**

Contractor: Research Triangle Institute
Contract Number: FQ761990044
Effective Date of Contract: 30 June 1989
Contract Expiration Date: 30 September 1990
Short Title of Work: DAHPHRS
Period of Work Covered: Jun 89 - Sep 90

Principal Investigator: Charlotte O. Scheper
Phone: (919) 541-7116

RL Project Engineer: Patrick J. O'Neill
Phone: (315) 330-4361

Approved for public release; distribution unlimited.

This research was supported by the Strategic Defense Initiative Office of the Department of Defense and was monitored by Patrick J. O'Neill, RL (C3AA), Griffiss AFB NY 13441-5700 under Contract FQ761990044.



Accession No.	
NTIS <input checked="" type="checkbox"/>	
DTAC <input type="checkbox"/>	
Unannounced <input type="checkbox"/>	
Justification	
By	
Distribution/	
Availability Codes	
Avail and/or	
Pist	Special
A-1	

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (Leave Blank)		2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
		December 1991	Final Jun 89 - Sep 90
4. TITLE AND SUBTITLE INTEGRATION OF TOOLS FOR THE DESIGN AND ASSESSMENT OF HIGH-PERFORMANCE, HIGHLY RELIABLE COMPUTING SYSTEMS (DAHPHRS)		5. FUNDING NUMBERS C - FQ761990044 PE - 63223C PR - 2300 TA - 03 WU - 06	
6. AUTHOR(S) Charlotte O. Schepers, Robert L. Baker, Harold L. Waters II			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Research Triangle Institute Center for Digital Systems Research Research Triangle Park NC 27709		8. PERFORMING ORGANIZATION REPORT NUMBER N/A	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Strategic Defense Initiative Office, Office of the Secretary of Defense Wash DC 20301-7100		10. SPONSORING/MONITORING AGENCY REPORT NUMBER RL-TR-91-397	
11. SUPPLEMENTARY NOTES Rome Laboratory Project Engineer: Patrick J. O'Neill/C3AA/(315) 330-4361			
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution unlimited.		12b. DISTRIBUTION CODE	
13. ABSTRACT (Maximum 200 words) Systems for Space Defense Initiative (SDI) space applications typically require both high performance and very high reliability. These requirements present the system engineer evaluating such systems with the extremely difficult problem of conducting performance and reliability trade-offs over large design spaces. A controlled development process supported by appropriate automated tools must be used to assure that the system will meet design objectives. This report will describe an investigation which examined methods, tools, and techniques necessary to support performance and reliability modeling for SDI systems development. Models of the JPL Hypercube, the Encore Multimac, and the C.S. Draper Lab Fault-Tolerant Parallel Processor (FTPP) parallel computing architectures using candidate SDI weapons-to-target assignment algorithms as workloads were built and analyzed as a means of identifying the necessary system models, how the models interact, and what experiments and analyses should be performed. As a result of this effort, weaknesses in the existing methods and tools were revealed and capabilities that will be required for both individual tools and an integrated toolset were identified.			
14. SUBJECT TERMS Fault-Tolerant Computing, Performance Evaluation, Multiprocessors, Reliability Analysis, Parallel Processing, CAE Tool Integration		15. NUMBER OF PAGES 236	
		16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL

Contents

List of Acronyms and Symbols	x
Acknowledgments	xii
1 Introduction and Executive Summary	1
1.1 Scope and Objectives	1
1.2 Problem Description	2
1.3 Approach	3
1.4 Conclusions	6
2 Methods and Tools	9
2.1 Design Methodology	9
2.2 DAHPHRS Methods	12
2.2.1 Modeling Phase Framework	12
2.2.2 Performance Analysis	21
2.2.3 Reliability Analysis	23
2.2.4 Fault Tolerance Analysis	28
2.2.5 Integrated Performance and Reliability Analysis	31
2.3 Required Tools	33
2.3.1 Application/Algorithm, Architecture Description, and Workload Characterization	33
2.3.2 Performance Modeling Tools	39
2.3.3 Reliability Modeling Tools	42

2.3.4	Summary	44
3	Baseline Determination Modeling Phase	46
3.1	Description	46
3.2	Baseline Determination Case Studies	49
3.2.1	Function Library	52
3.2.2	High-Level Performance Assessment	65
3.2.3	Network Reliability Analysis	71
4	Initial Design Modeling Phase	88
4.1	Description	88
4.2	Initial Design Case Studies	91
4.2.1	Improved Workload Characterization	93
4.2.2	High-Level Performance Assessment for Parallel Decomposition	99
4.2.3	Parametric and Phased-Mission Reliability Analysis	105
5	Design Refinement Modeling Phase	120
5.1	Description	120
5.2	Design Refinement Case Studies	124
5.2.1	Operating System Performance Modeling for Distributed Real-Time Systems	125
5.2.2	Behavioral Modeling for Fault Tolerance Evaluation	147
5.2.3	Reliability Analysis with Measured Parameters	187
6	Conclusions	192
References		199

List of Figures

1.1	Paradigm for Performance and Reliability Modeling in Support of System Development	4
2.1	Design-Cycle Milestones	11
2.2	System Requirements Review Actions	13
2.3	System Design Review Actions	14
2.4	Preliminary Design Review Actions	15
2.5	Baseline Determination Phase Process Diagram	17
2.6	Initial Design Phase Process Diagram	19
2.7	Design Refinement Phase Process Diagram	20
2.8	Performance Modeling Process	22
2.9	Reliability Analysis Process	24
2.10	Impact of Mission Characteristics on Reliability Determination	26
2.11	Models for Fault Tolerance Analysis	29
2.12	Integrated Model	30
2.13	Models for Integrated Performance and Reliability Analysis	32
2.14	Integrated Tools	34
3.1	Baseline Determination Relationship to Methodology	46
3.2	Baseline Determination Phase Process Diagram	48
3.3	Methodology for Use of Function Library	53
3.4	Function Library Structure	57
3.5	Shared Library Elements	58

3.6	Phase I Top-Level Performance Analysis	60
3.7	Phase II Top-Level Performance Analysis	61
3.8	Phase I TCD Component Performance Analysis	62
3.9	Phase II TCD Component Performance Analysis	63
3.10	Top-Level ADAS Graph of Row-by-Matrix Matrix Multiply Function	66
3.11	Non-Parallel ADAS Subgraph of Row-by-Matrix Matrix Multiply Function	67
3.12	Parallel ADAS Subgraph of Row-by-Matrix Matrix Multiply Function	68
3.13	Parallelism Recommendation Code Written in ADL	69
3.14	Eight-Node Hypercube Used for Example Solution	77
3.15	First Example FTPP System Analyzed	78
3.16	Second Example FTPP System Investigated	80
3.17	AOSP System	83
4.1	Initial Design Relationship to Methodology	88
4.2	Initial Design Phase Process Diagram	89
4.3	WAUCTION-ASSIGNMENT Procedure Call Diagram	94
4.4	Computational Model Performance Results	95
4.5	WAUCTION-ASSIGNMENT (Engineering Model) Measured Performance Results	95
4.6	Considered Cluster Interconnection Schemes	100
4.7	FTPP Configuration Studied	101
4.8	Sample FTPP Cluster Configuration	107
4.9	Phase I Model of 4×4 FTPP Cluster Configuration	109
4.10	FTPP Markov Model	111

4.11	Results of Permanent Processor Failure Rate Analysis	112
4.12	Results of Processor Recovery Rate Analysis	113
4.13	Results of Transient Processor Failure and Recovery Rate Analysis . .	114
4.14	Results of Network Element Failure Rate Analysis	115
4.15	Results of Variable Number of Triads Analysis	116
4.16	Engagement Phase Model	117
5.1	Design Refinement Relationship to Methodology	120
5.2	Design Refinement Phase Process Diagram	121
5.3	Application Software	127
5.4	Interprocessor Communication	128
5.5	Demonstration	129
5.6	Model One	131
5.7	Model Two	133
5.8	CLREAD.SWG	134
5.9	CLIENT.SWG	134
5.10	Server 0	135
5.11	Hardware Configuration	136
5.12	Software Design	138
5.13	512-Bit Message Throughput Results	140
5.14	16384-Bit Message Throughput Results	140
5.15	131072-Bit Message Throughput Results	141
5.16	512-Bit Message Server Utilization Results	142
5.17	16384-Bit Message Server Utilization Results	143

5.18	131072-Bit Message Server Utilization Results	143
5.19	512-Bit Message Network Utilization Results	144
5.20	16384-Bit Message Network Utilization Results	145
5.21	131072-Bit Message Network Utilization Results	145
5.22	Models for Fault Tolerance Analysis	151
5.23	Selected FTPP Configuration	153
5.24	Fork and Join Application	154
5.25	Fork and Join Application Model	156
5.26	FTPP Scheduler Model	158
5.27	Slave Timekeeper Model	159
5.28	Master Timekeeper Model	160
5.29	Network Services Top-Level Model	161
5.30	Network Services Send Message Model	162
5.31	Network Services Get Message Model	163
5.32	Fault Detection and Isolation Model	165
5.33	Processor Replacement a) Initial and b) Final Configuration Tables .	167
5.34	Processor Replacement Message Traffic	168
5.35	System Configuration before Total Reconfiguration	170
5.36	Final a) System Diagram and b) Configuration Table	173
5.37	Total Reconfiguration Network Element Message Traffic	174
5.38	Network Communication Frame	176
5.39	Cluster Level of FTPP Model	179
5.40	Network Element/FCR Level of FTPP Model	180

5.41 Processing Element Level of FTPP Model	180
5.42 Logical Fault Containment Design Error	182
5.43 Faulted Run Time Line	183
5.44 Triplex with FTPP Detection and Recovery States	188

List of Tables

2.1 Existing Tools	35
2.2 Architecture Description Information Requirements	36
2.3 Architecture Description Information Requirements	37
2.4 Application/Algorithm Description Information Requirements	38
3.1 Tools for Baseline Determination Phase	47
3.2 Summary of Phase I Case Studies for Baseline Determination Phase	50
3.3 Data Variables Used in the WTA/TS Algorithm	55
3.4 Parallelism Recommendation Matrix	70
3.5 Comparison of Low and High Estimates for Unknown Coefficient	77
3.6 Comparison of Bounds with Exact Reliability for 8-Node Hypercube	79
3.7 Reliability Bounds for Original Configuration of 4-Node, 24-Link FTPP	81
3.8 Comparison of Coefficients for Original and Alternative Configurations	82
3.9 Reliability Bounds for AOSP Network	86
4.1 Tools for Initial Design Phase	91
4.2 Summary of Phase I Case Studies for Initial Design Phase	92
4.3 Phase I and II WAUCTION-ASSIGNMENT Iterative Loop Behavior Values	97
4.4 Explored Mappings	102
4.5 Time Required for the Execution of the Parallel WTA/TS Algorithm (in seconds)	103
4.6 Dominant Algorithm Component (Utilization)	103
4.7 Phased Mission Analysis Results	118

5.1	Tools for Design Refinement Phase	123
5.2	Operating Systems Abstractions for System Services	127
5.3	Types of FTPP Network Traffic	152
5.4	Fork and Join Message Traffic Summary	155
5.5	Reconfiguration Messages and Actions	169
5.6	Total Network Element Traffic for Fork and Join Application	177
5.7	R_M , R_F , and R_R Results	190
5.8	C_F and C_R Results	190
6.1	Summary of Baseline Determination Case Studies	192
6.2	Summary of Initial Design Case Studies	193
6.3	Summary of Design Refinement Case Studies	193

List of Acronyms and Symbols

ACE	Array Computing Element
ADAS	Architecture Design and Assessment System
ADL	Attribute Definition Language
ADLEVAL	ADL Evaluator
AOSP	Advanced Onboard Signal Processor
ARIES	Automated Reliability Interactive Estimation System
ASSIST	The Abstract Semi-Markov Specification Interface to the SURE Tool
BBST	Bauer, Boesch, Suffell and Tindell
BM/C3	Battle Management/Command Control and Communication
CARE III	Computer-Aided Reliability Estimation
CASE	Computer-Aided Software Engineering
CDR	Critical Design Review
CFORM	Cluster Formation
CFRA	Contend for Recognition Authority
CRC	Cyclic Redundancy Check
CTU	Configuration Table Updates
DAHPHRS	Integration of Tools for the Design and Assessment of High-Performance, Highly Reliable Computing Systems.
DT	Demonstration, Evaluation and Test
EMI	Electromagnetic Interference
FC	Feasible Cluster
FDI	Fault Detection and Isolation
FDIR	Fault Detection, Isolation and Recovery
FMEA	Failure Modes Effects Analysis
FMGs	Fault Masking Groups
FT	Fault Tolerance
FTPP	Fault-Tolerant Parallel Processor
GLM	Gradiant, LaGrange Update and Median
GN	Global Notification
GNAs	GN Acknowledgements
HARP	Hybrid Automated Reliability Predictor

IOEs	Input/Output Elements
JPL	Jet Propulsion Laboratory
K-K	Kruskal-Katona
LAN	Local Area Network
LERP	Local Exchange Request Pattern
MIMD	Multiple Instruction Multiple Data
MIPS	Millions of Instructions per Second
NEs	Network Elements
OS	Operating System
PAWS	Padé Approximation with Scaling Program
PCSR	Processing-to-Communication-Speed Ratio
PDR	Preliminary Design Review
PE	Processing Element
PFCRs	Primary Fault-containment Regions
RA	Reconfiguration Authority
RCON	Reconfiguration
RTI	Research Triangle Institute
SDI	Strategic Defense Initiative
SDP	Sum-of-Disjoint Products
SDR	System Design Review
SERP	System Exchange Request Pattern
SRR	System Requirements Review
STEM	Scaled Taylor Exponential Matrix Program
SURE	Semi-Markov Range Evaluator Program
SYREL	Symbolic Reliability Algorithm
TCD	Target Cluster Definition
VHDL	VHSIC Hardware Description Language
VIDs	Virtual Group Identifications
VLSI	Very Large Scale Integrated Circuit
WAUCTION	Weapons Auction
WTA	Weapon-to-Target Assignment
WTA/TS	Weapon-to-Target Assignment/Target Sequencing
WTC	Weapon-to-Target Cluster Allocation

Acknowledgments

This research was sponsored by the Strategic Defense Initiative Office of the Department of Defense through the Rome Laboratory and the National Aeronautics and Space Administration's Langley Research Center under Contract NAS1-17964. The work was performed in the Center for Digital Systems Research of the Research Triangle Institute by C. Scheper (Project Leader), R. Baker, G. Frank, H. Waters, J. Bartlett, D. McLin, S. Mangum, A. Shagnea, A. Roberts, and E. Dashman. Joanne Bechta Dugan of Duke University investigated the applicability of analytical techniques for network reliability analysis. The research was directed by project technical monitors Patrick J. O'Neill and Capt. Gale Paeper of Rome Laboratory/COTC; and Wayne Bryant, Sally Johnson, and Carl Elks of NASA Langley Information Systems Division. The technical expertise and guidance provided by the Rome Laboratory technical monitors in space-based battle management systems and by the NASA Langley technical monitors in methods for evaluating fault-tolerant systems contributed significantly to this research. In addition Dr. Richard Harper and Carol Babikyan of Charles Stark Draper Laboratory, Inc. provided the fault-tolerant parallel processor design information which was the basis of the fault tolerance evaluation case study in this research and graciously answered the numerous questions regarding the processor design.

Technical support in preparing the DAHPHRS documentation and this report was provided by G. Loveland, J. Muller, and B. Taylor.

1. Introduction and Executive Summary

1.1. Scope and Objectives

This report documents the results of the second phase in a three-phase research program to define an effective methodology and an associated toolset to evaluate the performance and reliability of complex computing systems designed for space-based battle management applications. The research was sponsored by the Strategic Defense Initiative Office of the Department of Defense through Rome Laboratory and the NASA Langley Research Center under Contract NAS1-17964.

A design and assessment framework for fault-tolerant systems has been proposed in a working document of the SDIO Battle Management/Command Control and Communications (BM/C³) Processor and Algorithm Working Group [1]. This framework seeks to assure that appropriate fault tolerance mechanisms are designed into a system to handle the fault classes expected to be encountered over the lifetime of the system, and thus assure that the dependability goals specified for the system are met. This framework also seeks to assure that the system can be validated. An important part of this framework is the use of models during the design process to evaluate the effectiveness of the proposed mechanisms. This includes reliability models that incorporate the effects of fault detection, fault isolation, and system reconfiguration; system performance models that measure the performance impact of the fault tolerance mechanisms; and system functional models that can be used to determine the effectiveness of the fault tolerance mechanisms.

The objective of the Integration of Tools for the Design and Assessment of High-Performance, Highly Reliable Computing Systems (DAHPHRS) program is to develop an integrated set of performance and reliability tools capable of managing the complexity of designing advanced systems and robust enough to adapt to the inevitable technological advances. This development will be accomplished by establishing the modeling methods required to meet the objectives of the Working Group framework, determining both the nature and fidelity of design information required to build those models, and developing requirements for automated tools that can implement the modeling methods. As a result of this work, a basis will exist for the development of a standard set of procedures that will enable the system engineer to meet the challenge of designing, evaluating, and selecting architectures for advanced systems.

1.2. Problem Description

The Strategic Defense Initiative (SDI) concepts require the development of systems for complex space applications subject to demanding real-time computational resource requirements and very high reliability requirements. These applications range from highly regular signal and image processing to highly dynamic mission planning functions. They are characterized by large amounts of numerical processing, large volumes of data, iterative processes to determine optimal solutions and data base searches. In addition, real-time deadlines must be met on a continuing basis. The missions that these applications address may require extremely long system operating life without repair or servicing. They are also divided into phases composed of long periods of moderate activity followed by very short periods of high activity and characterized by vastly different reliability and performance requirements. The system may operate under demanding weight and power requirements in an environment subject to radiation, thermal, and mechanical stress. These system characteristics lead to complex, nonuniform architectures consisting of a large number of processors with mechanisms to extend operational life and to support both changes in mission phase and attrition of system resources. Therefore, the applications require architectures that combine elements of massively parallel computing, reliability, fault tolerance, security, and safety.

Parallel computing is a rapidly emerging field in which many complex practical problems in realizing effective architectures have not been solved. Parallel systems are distinguished by the need to communicate between functions operating on multiple processors. This fact drives much of the performance and fault tolerance concerns. To achieve performance, communication mechanisms must be efficient. Fault tolerance in a parallel system dictates fault-tolerant communications, synchronization over multiprocessors, and global recovery from faults. The system mechanisms that provide this fault tolerance underlie the ultimate reliability of the system. An evaluation of an architecture would not be complete unless these features were assessed for effectiveness, completeness, and viability. The added complexity of providing fault tolerance presents the system engineer with the extremely difficult problem of conducting performance and reliability trade-offs over large design spaces and verifying performance and reliability over a wide range of operating conditions.

Additionally, the use of highly reliable, fault-tolerant systems in mission- and life-critical applications requires that their reliability and fault tolerance be validated. Life testing to establish reliability is not feasible due to very long mean times to failure. In fact, technological advances are producing systems of such complexity that it is not adequate to consider validation only after the system has been developed. Thus, there is a critical need for a comprehensive and uniform set of effective procedures to describe, model, and evaluate these systems. Such procedures would enable

aggressive consideration of reliability and fault tolerance requirements and their validation to be an integral part of the entire development cycle. These procedures are particularly needed in the early requirements and design concept stages, since it is in these development phases that uncovering design errors can have the greatest impact on system development costs.

The methods and tools necessary for the development of such procedures do not exist. This is due in part to the complexity of the systems; however, the very nature of these systems dictates evaluation criteria that differ in many respects from those used to evaluate more traditional computing systems and requires the evaluation of design concepts within a total systems framework. The evaluation must address: 1) the characteristics and requirements of the mission and environment in which the system will have to operate, as well as the specifics of the application software and the architecture; 2) the workload of the operating system, the communications, and the fault tolerance mechanisms, as well as that of the applications software; 3) the reliability and fault tolerance as well as the performance of the system; and 4) the factors that underlie reliability such as testability and maintainability.

1.3. Approach

The attributes of dependable computing systems include maintainability, availability, system integrity, validatability, complexity, and testability as well as performance and reliability, and there are many system assessment techniques appropriate to the different life-cycle phases, including formal methods, system testing, modeling, and simulation. To establish methods that can be used to meet the objectives of the proposed Working Group methodology, the scope of the DAHPHRS effort has been the use of analytic and simulation models to assess performance and reliability/fault tolerance. Since procedures are particularly needed in the early requirements and design concept stages, the DAHPHRS effort has been directed toward modeling methods appropriate to those stages. The overall DAHPHRS program was structured in three phases: the first two to establish the modeling methods and tool needs and the third to specify and build the necessary tools.

To establish the modeling methods and to determine what tools are needed and how the tools should interact, Phase I of DAHPHRS was directed toward relating current tools to the Working Group methodology framework and to the development and analysis of baseline system descriptions [2]. In Phase I, a paradigm of the design process for a Strategic Defense Initiative (SDI)-like system was developed. This paradigm was used to determine what system models are needed, how the models interact, and what experiments and analyses are needed. It was also used to illustrate how tools support the modeling methods and what the tool features and capabilities

should be.

In this paradigm, illustrated in Figure 1.1, the system development phases from system concept to implementation and test are carried out in the appropriate system engineering domains under the guidelines of the methodology. As architectures and

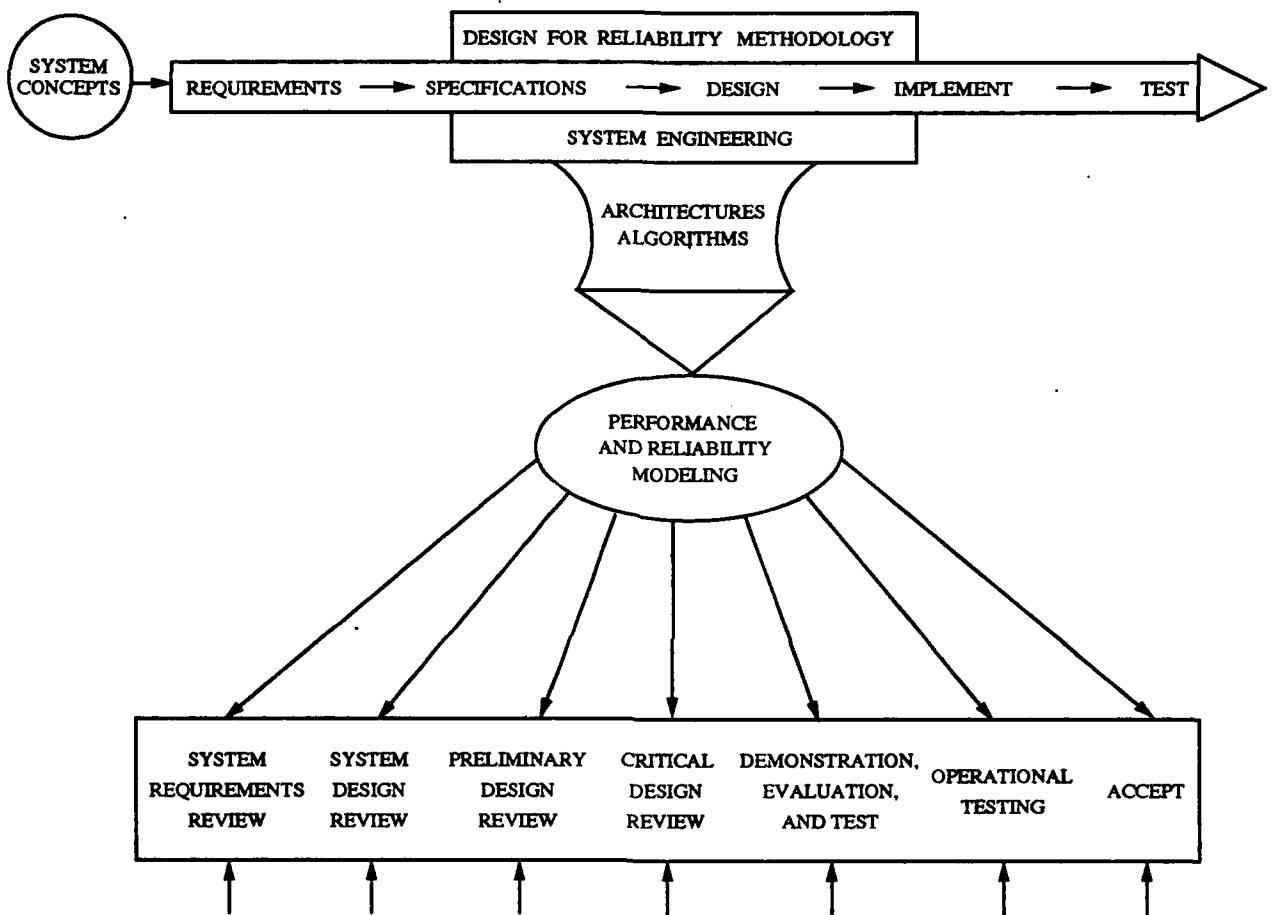


Figure 1.1. Paradigm for Performance and Reliability Modeling in Support of System Development

algorithms are developed in this process, performance and reliability tools assist the individual designers in evaluating and changing the designs and in maintaining the consistency of the designs with the overall system requirements and specifications. Selected results from the modeling effort are used to satisfy portions of the requirements for the various development milestones such as design reviews.

Since it was not possible to exercise and assess all aspects of performance and reliability modeling, effort was focused on those facets of the paradigm likely to reveal weaknesses in the existing methods and tools or likely to yield payoffs in the form

of refinements to large portions of the methodology. To this end, areas where the characteristics of the complex space mission are distinguished from more ordinary applications were considered of special interest, and two algorithms and three architectures were selected for analysis in Phase I. A typical SDI application could have requirements to detect and track potential targets and to allocate weapons necessary to destroy targets. The signal/image processing algorithms that would be employed to provide target detection, classification, tracking, and trajectory estimation are computationally intensive. However, most often they can be decomposed into highly regular computational structures that can be effectively handled by vector and pipeline processing techniques. The optimal allocation of weapon resources to targets requires the use of algorithms that differ significantly from signal/image processing algorithms. These mission planning functions employ linear, integer, nonlinear, or dynamic programming techniques with computational requirements that depend on the incoming target data and that vary with the number of targets. These algorithms are more difficult to decompose and embed in a parallel computing architecture and were therefore judged to be of particular interest for this effort. Two mission planning algorithms developed by Alphatech, Inc. [3] were selected and used in this study.

Meeting the demanding throughput requirements of such applications will require advanced architectures consisting of a large number of interconnected processors or computers. Of the various parallel computing architectures that have been proposed, three were selected for this study. The JPL Hypercube, an MIMD distributed-memory architecture, was selected primarily because the hypercube is one of the most extensively investigated parallel computing architectures. The Encore Multimax, an MIMD shared-memory architecture, was selected to provide contrast to the hypercube particularly in the area of interprocessor communications. Finally, a prototype version of the Fault-Tolerant Parallel Processor (FTPP) [4] being designed by Charles Stark Draper Laboratory was selected because it is the only parallel processing architecture that has the advanced fault-tolerant features necessary to attain very high reliability. As such, modeling for the FTPP is distinguished from ordinary parallel processor modeling and should be expected to provide insight into weaknesses in the methods and tools as they pertain to high-reliability applications.

In Phase I, emphasis was placed on the impact of parallel processing issues on the methods and tools for performance analysis, with some aspects of fault tolerance drawn into the performance modeling and some high-level reliability analysis conducted. In Phase II of the DAHPHRS program, which is documented in this report, some areas of performance modeling techniques were investigated further, but the primary emphasis was on modeling fault-tolerant mechanisms and reliability analysis. The Phase I case studies identified four areas in performance modeling for additional study in Phase II: stochastic attributes, operating system models, a function library, and high-level parallel decomposition techniques. In addition to the more detailed reliability analyses undertaken in Phase II, a survey of analytic methods for network

reliability analysis was undertaken.

Since the primary objective of Phase II was to investigate means of modeling the fault tolerance mechanisms of a system, the FTPP was selected for the case studies as the only fault-tolerant architecture of the three from Phase I. The level of detail at which the fault tolerance mechanisms were modeled necessitated access to design information for fault diagnosis and system reconfiguration algorithms. This information is nonexistent or difficult to obtain for many proposed architectures, but was provided by Draper Laboratory for the FTPP.

With the completion of the Phase I and II case studies undertaken for the paradigm, the established modeling methods were compiled into three phases of integrated analysis methods to address the different needs of the early- to mid-design phases. From this compilation, the required types of tools were identified and an approach to their integration was proposed. With the completion of Phase II, requirements can be specified for the integration of existing tools with a common data base and experiment manager.

1.4. Conclusions

During the two completed phases of the DAHPHRS program, selected portions of the Working Group Methodology framework for the development of dependable systems have been instantiated and carried out. Through the use of case studies, the following has been accomplished: 1) Demonstrated the integration of performance, reliability, and fault tolerance evaluations to effect critical design decision trade-offs 2) Established the need for and demonstrated the value of models which combine effects and capture interactions of the architecture, operating system, application software, communication system services, and the hardware and software fault tolerance mechanisms 3) Demonstrated the need for behavioral/functional modeling to support fault tolerance evaluation 4) Identified the types of analyses needed to support the development of complex systems. In addition to the modeling techniques that were identified and demonstrated, information requirements for model development for early and mid-development phases were established, model fidelities for varying degrees of model refinement were demonstrated, the need for measurement to support modeling was demonstrated, and the need for model validation was established.

This work has established the need for and demonstrated the value of using models to address system behavior in the early stages of system development. We have defined a modeling framework, constituent modeling processes, and integrated evaluation processes. The major components of this framework are the system performance/fault

tolerance model and the system reliability model. Three modeling processes are associated with these components: performance analysis, fault tolerance analysis, and reliability analysis. Each process defines the types of models and evaluations required to conduct the associated type of analysis. Given the modeling framework and the constituent modeling processes, integrated evaluation processes can be defined tailored to specific stages of the overall system design process. We have defined three such evaluation processes: Baseline Determination, Initial Design, and Design Refinement.

The current tools were found to have limitations in their ability to handle the size and complexity of parallel, fault-tolerant architectures for SDI algorithms. In particular, the capability to create and simulate the system behavioral model required to evaluate fault-tolerant performance of a parallel architecture does not exist in current tools to a sufficient level. Further, the current tools were built to address performance and reliability analysis separately. One of the major problems encountered in carrying out experiments in the case studies was the management and effective analysis of the outputs from the simulations and analyses. Effective design tools are needed that allow simulatable, hierarchical models to be created at the outset and refined throughout the design cycle. These tools must also be capable of managing and analyzing large experiment outputs so that the user can effectively carry out evaluations and make design decisions.

The successful development of a useful, fully integrated toolset depends on the degree of maturity of its underlying methodology. A stepwise development will allow the experimentation and practical application that will assure a mature methodology. This development can start now by integrating existing tools with a common data base and experiment manager, and proceed by making necessary modifications to existing tools, culminating in new tools that can provide the full range of capabilities the methodology requires.

There is also a critical need to consolidate the methods into a comprehensive set of procedures by which distributed, fault-tolerant system architectures can be evaluated in the early- to mid-design stages. Such evaluations would support the selection of candidate designs from a larger set of competing candidates and would identify critical design risk areas for the purposes of reducing overall technical risk and avoiding the commitment of implementation resources to unsuitable designs. The objectives of establishing such procedures would be to reduce the latest evaluation techniques to practice, to establish a common basis for design descriptions, to establish a common basis for evaluating systems, and to make the first step toward Department of Defense (DoD) standards for system evaluation procedures and guidelines.

With the completion of Phase I and Phase II of the DAHPHRS program, modeling methods for implementing the performance, reliability, and fault tolerance evalua-

tions for the early- to mid-design phases of the Working Group design framework have been established. It has been demonstrated that although fault tolerance evaluation is usually done on a prototype of the architecture, it can be pushed back into earlier design phases by the use of modeling techniques that can integrate appropriate components of system behavior and architecture. The value of these techniques was demonstrated by uncovering design flaws and limitations in a fault-tolerant system currently under development. The establishment of these modeling methods forms the basis for developing an integrated toolset to support the design framework. It also forms the basis both for extending the methods to include other attributes of dependable processing and to address other stages of system development and validation and for consolidating the methods into a comprehensive set of evaluation procedures.

2. Methods and Tools

2.1. Design Methodology

The dual requirements of high reliability and high performance for systems that will operate nearly autonomously in mission- and life-critical applications dictate that those systems be validated to a high level of confidence. Accordingly, it is expected that a rigid development process be utilized to assure that design errors are eliminated before the system is delivered and to assure that the system will meet reliability and performance objectives. The "design-for-reliability" methodology framework that has been set forth in the working document of the SDIO BM/C³ Processor and Algorithm Working Group [1] specifies an iterative process consisting of the following seven steps:

1. identify classes of expected faults over the lifetime of the system
2. specify goals for the dependability of system performance
3. partition the system into subsystems for implementation, taking into account both performance and fault tolerance
4. select error detection and fault diagnosis algorithms for every subsystem
5. devise state recovery and fault removal techniques for every subsystem
6. integrate subsystem fault tolerance on system scale
7. evaluate the effectiveness of fault tolerance and its relationship with performance

The design is refined by iteration on steps three through seven.

Five phases of design are specified and milestones established for each phase. Each phase is concluded with a review, as summarized below:

1. system requirements review (SRR) to specify a computational model, requirements for performance and fault tolerance, applicable architectural approaches, and a development plan
2. system design review (SDR) to evaluate architectural trade-offs and to select an architectural approach and fault tolerance strategy
3. preliminary design review (PDR) to specify preliminary hardware and software design and to provide performance and fault tolerance evaluation

4. critical design review (CDR) to provide a completed hardware and software design, refined analysis of performance and fault tolerance attributes, and a plan for a feasibility demonstration
5. demonstration, evaluation, and test (DT) review to include a demonstration of brassboard components and operational software, and an experimental evaluation of performance and fault tolerance features

Figure 2.1 illustrates the sequence of design-cycle milestones and the deliverables for each milestone.

The framework stresses the use of models to evaluate proposed designs and the need for tools at all levels of the design process. During the system requirements phase, tools are required to evaluate very high-level designs without detailed hardware and architectural information. These tools must interface with the tools that analyze more detailed designs in later phases. The outputs of these tools should be usable as inputs to more detailed tools and/or should be compared with more detailed evaluation results. These interface capabilities are necessary to verify that the high-level requirements are met by the actual design. It should be possible for tools to share data files at all levels in the design process.

During system design, when architecture alternatives are evaluated, the framework requires tools that will allow meaningful comparisons of the performance and fault tolerance attributes of each alternative system. The tools must model high-level architecture features and must incorporate a high-level fault model. In the Working Group report, it is also recommended that error propagation effects and the effects of corruption of system state due to faults be modeled at this level. An accurate testability analysis is also required.

During the preliminary design phase, more details of the selected design are available, and the framework calls for tools whereby models created for the system design phase can be refined to reflect the newly available detail. More accurate estimations of performance and fault tolerance parameters should be possible in this phase, and the tools should now be able to provide accurate estimates of coverage of the error detection mechanism and to evaluate the quality of the error containment and error recovery procedures. The tools must have a clean interface to the more detailed tools that will come later and to the more general tools used earlier. Accurate reliability modeling tools are also a necessity at this stage.

By the critical design review, details of both hardware and software designs are completed. The evaluation tools should allow further refinement of the models to reflect the additional detail and should allow detailed hardware and software simulations to be performed. Since existing tools cannot handle the complexities of modern designs

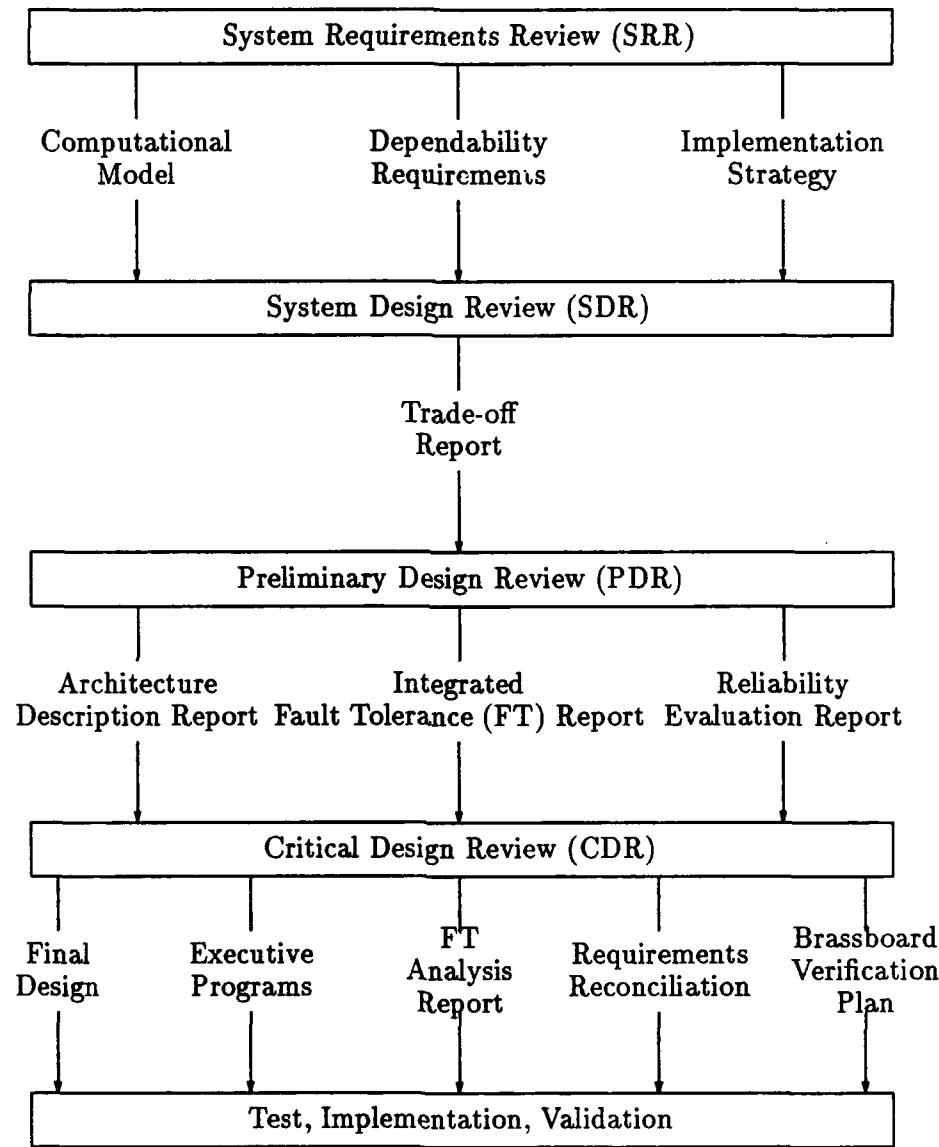


Figure 2.1. Design-Cycle Milestones

at this stage, the Working Group concludes that some form of hierarchical simulation will be needed, whereby a small part of the system can be modeled in great detail and interfaced with higher-level simulations of the remainder of the system. This will require a clean interface between the higher-level and the lower-level simulation tools. Reliability models will also need to be refined to reflect the more detailed information. Simulation results, such as coverage factors, recovery times, etc., need to be easily transferred from the simulation program to the reliability analysis program.

At the test, implementation, and validation review, the results of experiments performed on the prototype would be presented. The reliability modeling that began with high-level models in the early design stages and culminated in the detailed models required at the critical design review should have predicted a behavior of the system that can be verified by actual injection experiments. The modeling and simulation tools should be interfaced with the testbed so that the required inputs to the testbed can be generated automatically and the outputs compared with those predicted by the simulations.

The Working Group concludes that the accomplishment of the objectives of each of the design phases for large, complex computing systems requires a hierarchical model of the system, and that the iterative nature of the design process requires that the set of tools based on that model be integrated to maintain consistency in the data and descriptions of the system.

2.2. DAHPHRS Methods

2.2.1. Modeling Phase Framework

Given the Working Group methodology framework and its reliance on tools to support the design process, the objective of Phase II of the DAHPHRS program was to establish the modeling methods that underlie the requirements for integrated tools to support that framework for performance and reliability analysis. These methods target the early- to mid-design phases that correspond to the system requirements review, the system design review, and the preliminary design review. Figures 2.2, 2.3, and 2.4 illustrate the major activities that were specified by the Working Group to accomplish the deliverables for the SRR, SDR, and PDR milestone reviews.

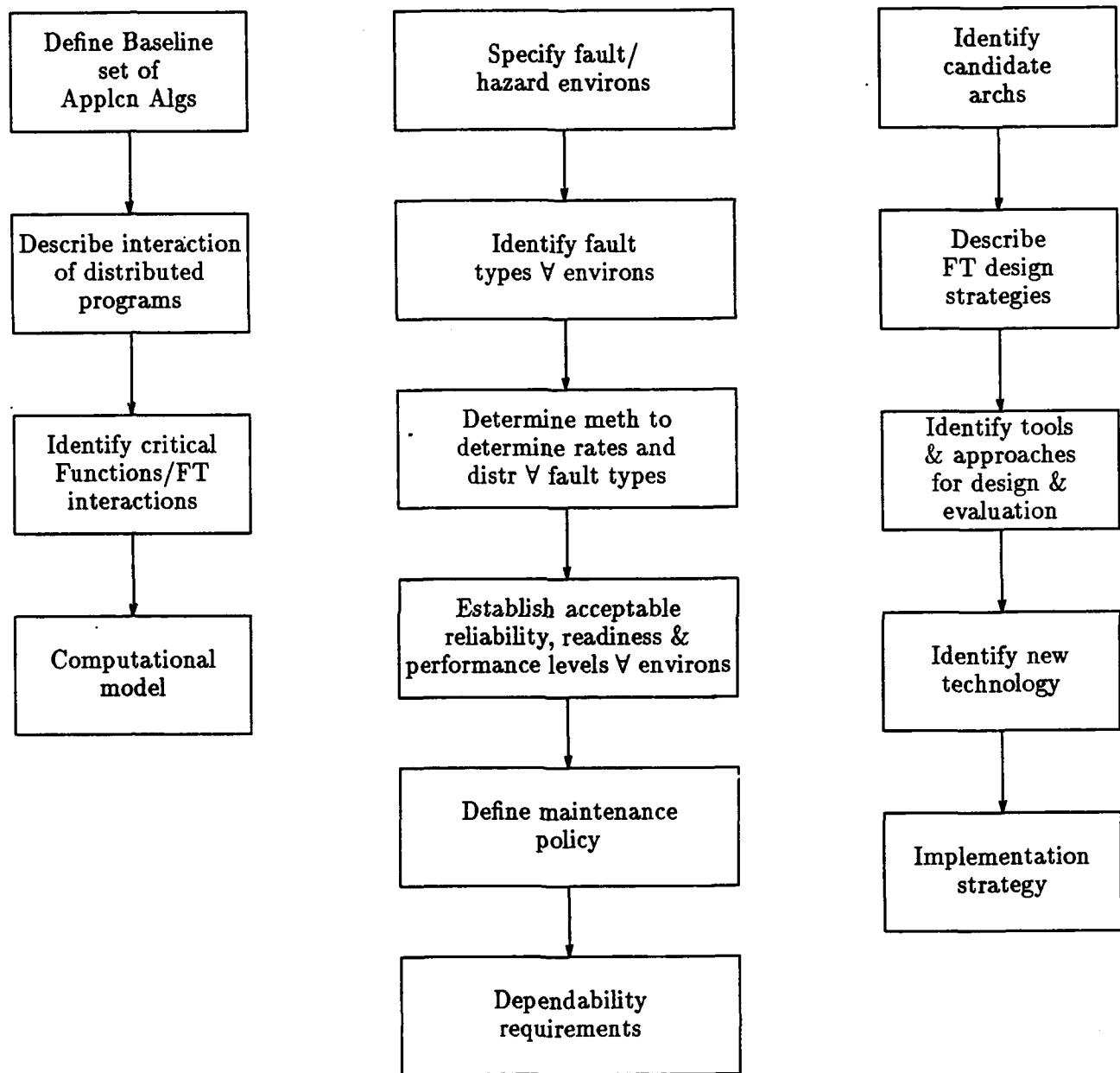


Figure 2.2. System Requirements Review Actions

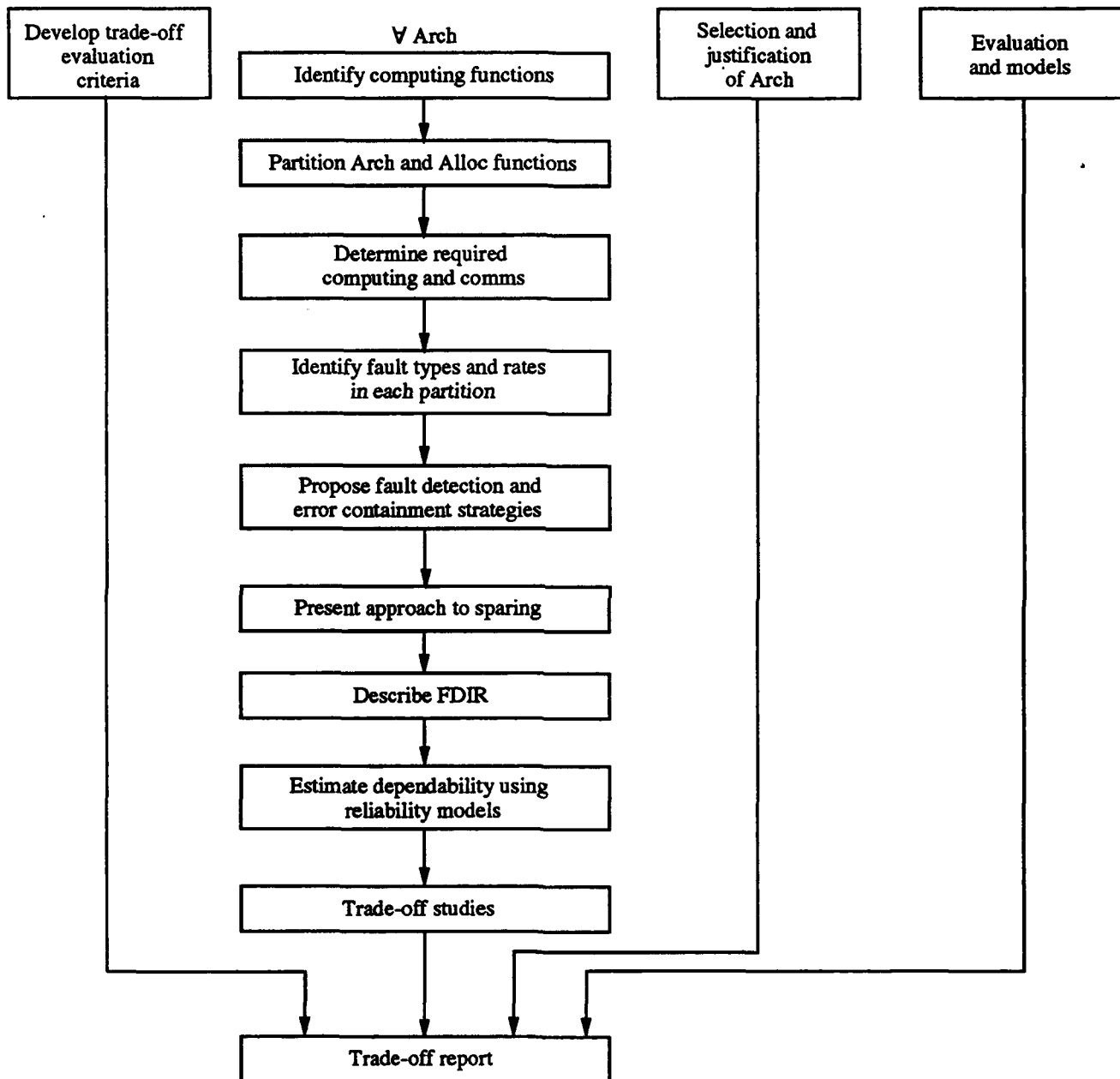


Figure 2.3. System Design Review Actions

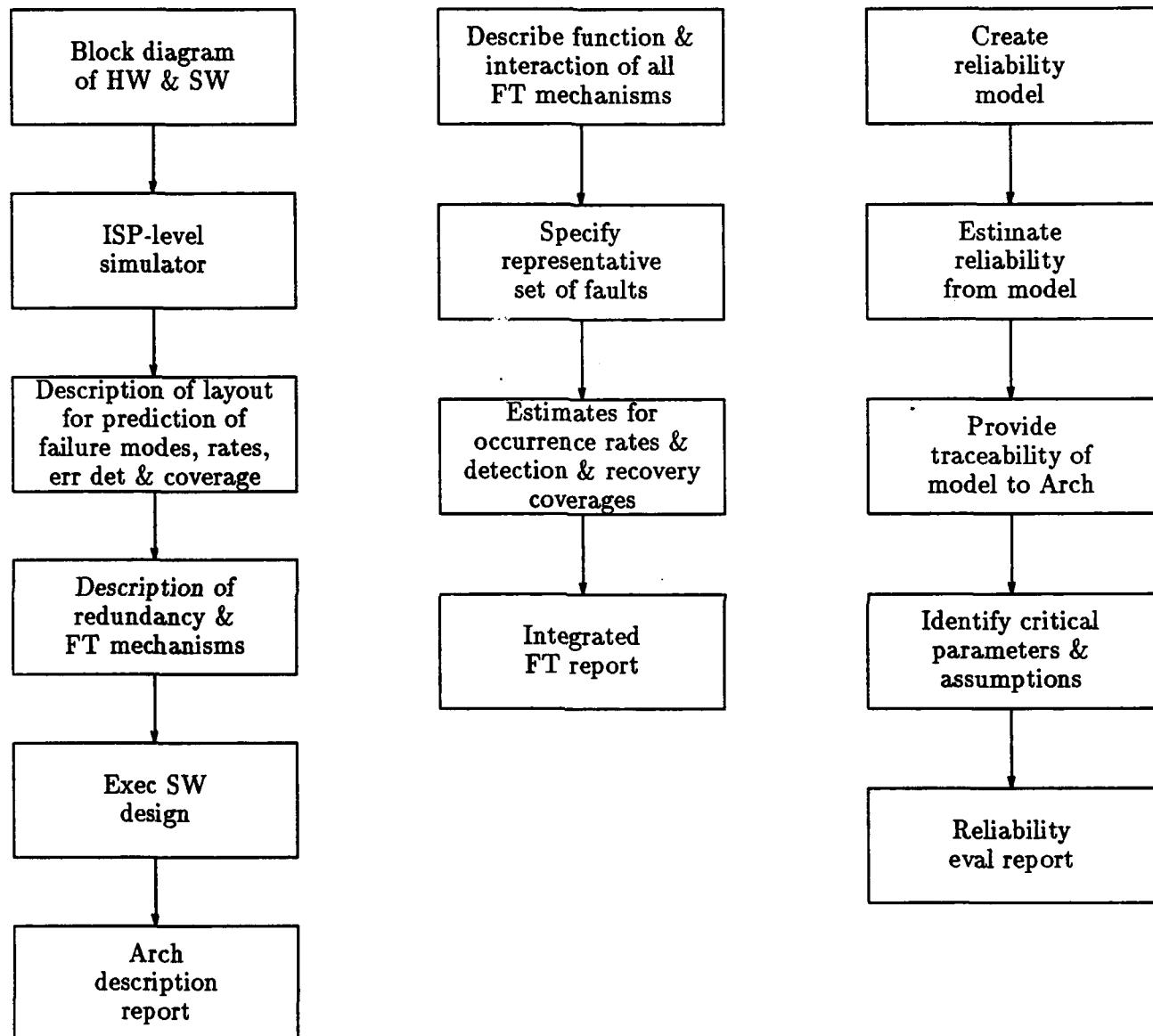


Figure 2.4. Preliminary Design Review Actions

To support the iterative nature of the design process and to address the different needs of the design-cycle milestones specified by the Working Group, the DAHPHRS methods have been structured into three phases: baseline determination, initial design, and design refinement. Each phase relies on the availability of a specified level of information from which particular system evaluations can be made. Each phase assumes an iterative process of model construction, analysis, and refinement until the design space has been examined sufficiently to determine if a design can be expected to meet the system requirements. The measures that can be accomplished in a particular phase of modeling are dependent on the level of detail available for the components of the design. As the design progresses and more detailed information becomes available, the system engineer can refine the models developed at a previous phase and focus the evaluations to produce the measures needed to proceed with the design. The definition of the phases establishes the relationship between the level of information available and the level of modeling that can be accomplished, and thus establishes the measures and trade-offs that are possible.

In keeping with the aims of the Working Group methodology, the methods stress the consideration and assessment of reliability and fault tolerance from the very beginning of the design. If a system must meet demanding performance and reliability requirements, it cannot be accurately assessed by independent analysis of its performance and reliability. Mechanisms to assure fault tolerance will have to be included for the system to attain its reliability requirements. The efficiency and efficacy of these mechanisms must be addressed in the system performance analysis as well as in the reliability analysis since they require significant processing resources and have to be executed within strict timing constraints. Furthermore, the definition of what constitutes an operational versus a failed system state for reliability analysis has to include an assessment of the ability of the system in that state to achieve the required performance levels. Therefore the DAHPHRS methods include techniques for using interactions between models to achieve an integrated analysis of performance and reliability/fault tolerance.

The process and information flow of the baseline determination phase is illustrated in Figure 2.5. It is in this phase that the basic architectural and algorithmic structures and characteristics are defined and their ability to meet the requirements of the application and mission evaluated. This phase corresponds to the first iterations through the design steps where the identification of fault sets, the specification of performance and dependability goals, and the system partitioning are of prime importance. The application and mission requirements are the only data required for baseline determination. From this data, a computational model can be constructed and the capacity of different architectural configurations to meet the application performance and reliability requirements can be evaluated.

At the end of the baseline determination, the system engineer will have high-level

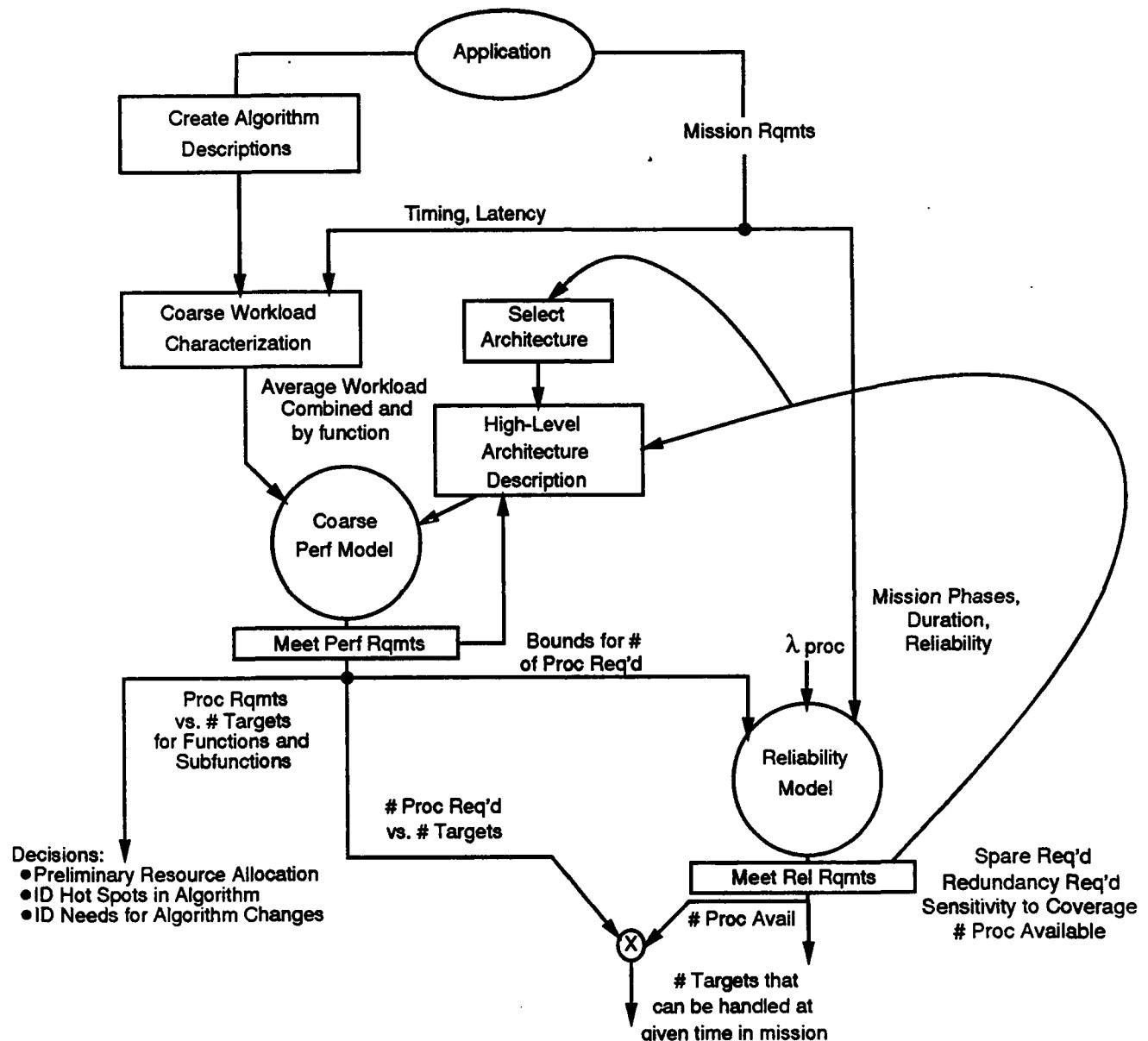


Figure 2.5. Baseline Determination Phase Process Diagram

architectural designs and preliminary resource allocations for those designs. He can make estimates of processing requirements for the various application functions in terms of mission parameters and estimates of available processing resources at given mission times. He can determine the number of processors required for performance and the number required for reliability. He can identify potential hot spots and changes in algorithms and can establish the sensitivity of system reliability to overall coverage of faults.

When the baseline features of the design have been established and initial assumptions can be made regarding the fault detection, isolation, and recovery mechanisms (FDIR), as specified by steps four through six of the Working Group methodology (see page 9), the baseline determination results can be consolidated into a level of design sufficient to select the architecture or architectures to be carried forward. This initial design phase, illustrated in Figure 2.6, requires the construction of a more detailed computational model from the baseline requirements computational model and algorithm descriptions. This refined model must address the rules for decomposing the algorithm into parallel and sequential components and for judging the appropriate granularity for the parallel components. It also must provide a breakdown of workload into processing and communication components. From the initial design analysis, the system engineer can evaluate the sensitivity of system performance to architectural parameters and resource allocation and the sensitivity of system reliability to the performance of the proposed fault tolerance mechanisms and assumed failure modes.

The design refinement phase, illustrated in Figure 2.7, allows a more detailed analysis of the candidate designs derived during the initial design. The models and analyses in this phase particularly focus on the fault tolerance mechanisms since the design has proceeded far enough that sufficiently detailed information is available to model those mechanisms. This level of analysis also requires that algorithms and operating system software be designed to a level of detail sufficient to gauge their interactions with the fault tolerance mechanisms. It is also at this stage that the system parameters most important to attaining reliability requirements can be identified and estimated. The design refinement analyses result in more accurate design measures and in time estimates and functional evaluations of FDIR mechanisms.

The general requirements and techniques for performance, reliability, and fault tolerance evaluation within the DAHPHRS modeling phase framework are discussed in the following sections. Specific issues and case studies for each of the three phases are discussed in the succeeding chapters of this document.

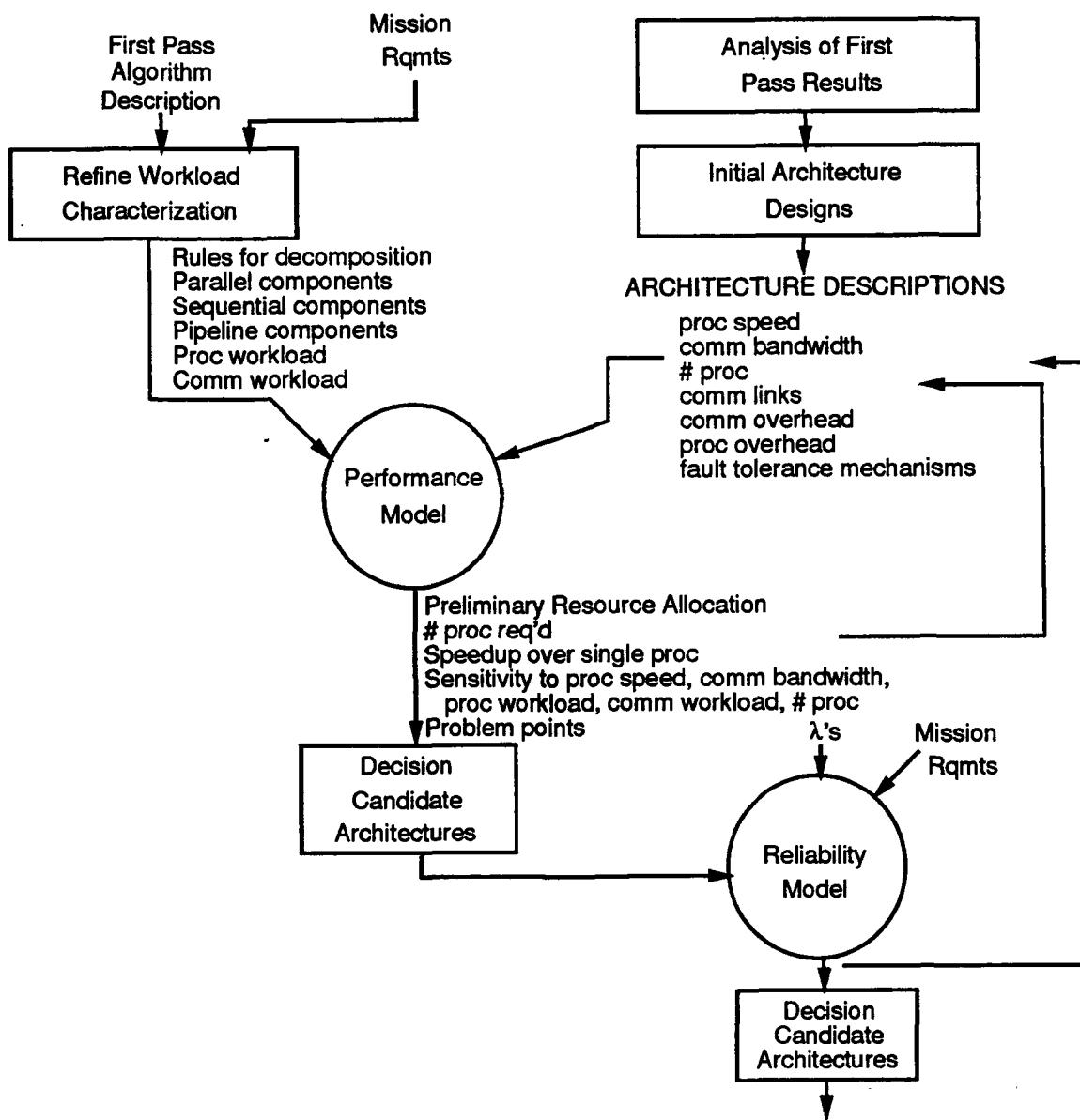


Figure 2.6. Initial Design Phase Process Diagram

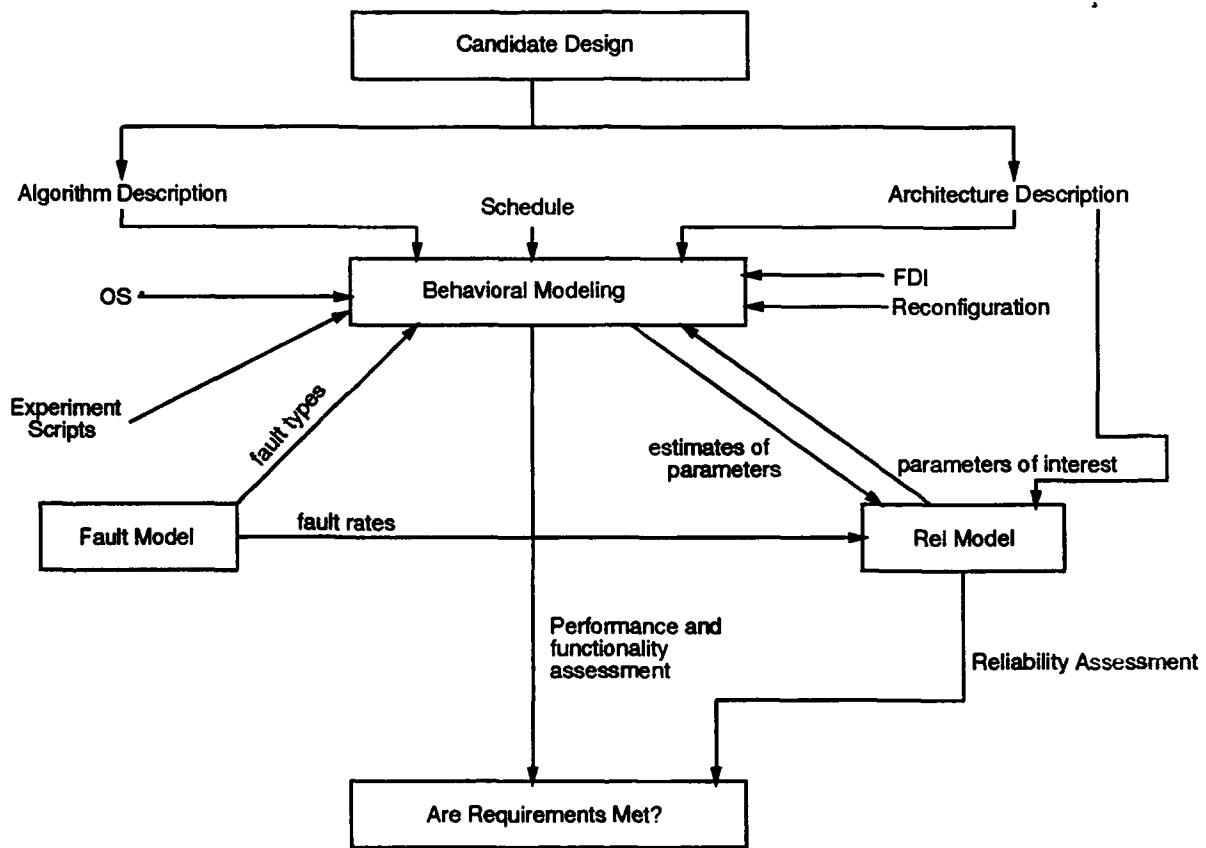


Figure 2.7. Design Refinement Phase Process Diagram

2.2.2. Performance Analysis

The goal of performance analysis is to evaluate the capacity of the proposed architecture to meet the range of requirements specified for processing, memory, and communications. To accomplish this goal, performance analysis must evaluate resource utilizations, processing workloads, and I/O and interprocessor communication workloads. It also must identify "bottlenecks" or "hot spots" in the architecture. The performance of these evaluations throughout the design cycle requires the capability to produce coarse to refined performance estimates based on workload characterization and architecture description. These estimates must provide bounds on workload throughput, utilization of resources by application, the effect of any contention for resources on compute time and utilization, the sensitivity of performance to architectural parameters and resource allocation, and performance measures of fault tolerance mechanisms.

System performance can be modeled by multiple models of increasing detail and complexity consistent with the amount of information available at particular design stages. Consequently, the methodology requires a decision as to what level of model is needed to support the required level of analysis for a given architecture at a given design phase. At the highest level, the system is modeled by an analytical model using primitive information such as processing and communication workloads and I/O and memory bandwidths that are required for the architecture to support the application algorithm. As the hardware and software of the system are defined in more detail, performance modeling by simulation and engineering models can be initiated. The performance modeling process, as illustrated in Figure 2.8, starts with a description of the architectures and algorithms that make up the system. From this description, parameterized attribute, functional, and engineering models can be developed.

The parameterized attribute model is the basic performance model. It consists of a data/control flow model of the algorithmic processes which has been mapped, or constrained, to a structural model of the architecture. The functional model describes the behavior of either the components of the architecture or the algorithms that will be executed by the system. Finally, the engineering model consists of system prototypes for the algorithms and/or architectures. All three models produce performance predictions, the attribute and functional models through simulation and the engineering model through measurement. The three models interact through parameter values computed by the functional and engineering models for use by the attribute model. Each of the models can be built to whatever degree of detail is reasonable for the stage of design under consideration. The cross-validation that occurs among the models as they produce consistent performance assessments allows the models to be used and built upon with confidence at subsequent levels. Thus, performance models that have been built from and validated by measurements from engineering

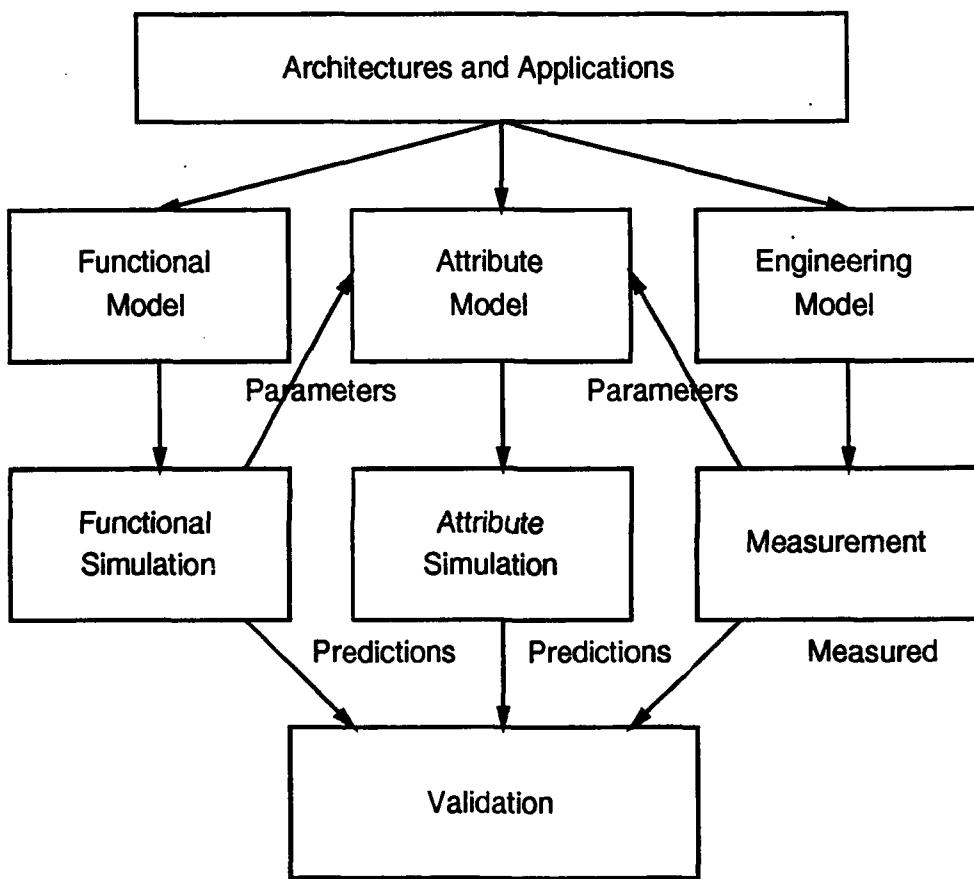


Figure 2.8. Performance Modeling Process

models can be used in later analyses or subsequent system designs without the need for implementing a full engineering model.

2.2.3. Reliability Analysis

The goal of reliability analysis is to evaluate the probability of successfully completing a mission given that faults will occur as defined by the fault models specified for the mission and the system implementation technology. It entails assessing the proposed redundancy and sparing levels for the architectural elements for all mission phases and determining the sensitivity of the analysis results to variations in parameters used in the analysis, such as assumed failure rates and rates and effectiveness of fault detection, isolation, and recovery (FDIR) processes. The effort required to determine the reliability of a system to an acceptable level of confidence is driven to a large degree by the per-hour probability of failure. However, other factors contribute to the difficulty of establishing system reliability, including mission duration, system complexity, and characteristics of the failure processes. Therefore, the overall mission types and requirements, the mission environment, the fault-tolerant system characteristics, and the technology of implementation have to be considered in determining appropriate methods for particular reliability regimes. Mission environment and implementation technology dictate the failure processes and modes that must be tolerated. Thus, the fault model by which system reliability is judged is of fundamental importance and must account for all relevant physical failure mechanisms, as well as design faults and software failures resulting from system design and implementation.

The reliability analysis process, as illustrated in Figure 2.9, starts at the mission requirements level and incorporates the system architecture, the technology of implementation, and the appropriate failure processes.

From here, the process proceeds to establishing reliability determination requirements. These requirements specify the type of analysis required to produce a rigorous, statistically well-founded reliability determination for the type of system, type of mission, and range of reliability required. They will also specify the type of experimentation procedures required to support that analysis. In particular, these requirements will specify the combination of methods and tools that should be used, identify the modeling assumptions, specify how the assumptions will be supported, specify the model validation requirements, specify the fidelity requirements for model parameters, and specify how the parameters will be determined.

The level of confidence that can be placed in reliability predictions and measures varies with the accuracy of the data and the suitability of the techniques used to derive them. Both of these factors are functions of the design-cycle stage at which the determination

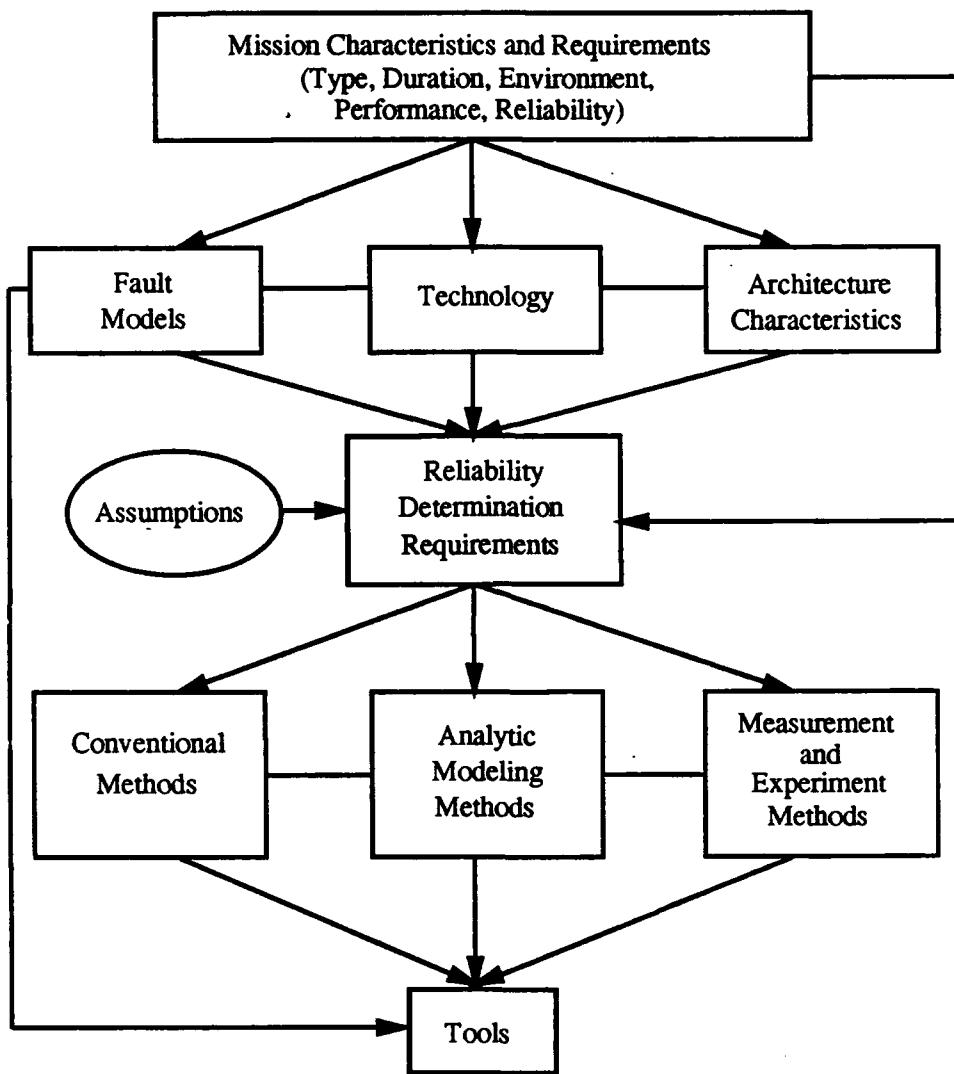


Figure 2.9. Reliability Analysis Process

is made, of the overall level of reliability required of the system, and of the sensitivity of the system reliability to the performance and functionality of the fault tolerance mechanisms and assumed failure modes. As the fault tolerance of a system is increased to attain the ultra-high levels of reliability desired for mission- or life-critical systems, the task of determining system reliability becomes more difficult because it becomes more sensitive to the performance and functionality of the fault tolerance mechanisms and the validity of the assumed failure modes. System reliability becomes more dependent on the timely execution of mechanisms that are difficult to quantify or verify and system failure is more likely to be caused by rare external events that are not easily incorporated into existing models (e.g., correlated faults, EMI effects). Additionally, these transitions from one system state to another may not fit the statistical assumptions of existing models and their statistics may not be readily available. Therefore, establishing reliability determination requirements is basic to selecting methods and tools that are well matched to the systems they will be used to evaluate. Otherwise, it would be easy to produce meaningless predictions from models that do not accurately describe the system behavior and tools that were designed for different system assumptions. Figure 2.10 illustrates the relationship of mission characteristics of a hypothetical space satellite system to assessment factors that determine the reliability determination requirements for the system.

Credible reliability determination should address the problem of system failure from a complete range of perspectives, including hardware design faults, communications network reliability, software errors, and man-machine interface faults as well as hardware operational faults. However, current analytic modeling techniques are limited in the area of software design faults. Software reliability models have not achieved the degree of credibility associated with hardware models. Also, more work needs to be done in identifying interactions of software states and hardware failures that lead to system failure.

The reliability determination methods necessary for SDIO BM/C³ applications include analytic modeling and experimentation methods. Conventional methods for establishing the reliability of systems and components use a combination of techniques such as life testing, accelerated life testing, use of service data, accepted guidelines such as MIL-HDBK-217 based on verified theoretical models and experience, failure modes effects analysis (FMEA), fault tree analysis, and quality control. These techniques used in combination have been adequate to establish the reliability of moderately complex systems and even relatively simple redundant path systems and can be used to establish the reliability of components of a redundant system. However, for highly reliable fault-tolerant systems, conventional methods are not adequate. For example, life testing sufficient to establish a high level of confidence may require testing thousands of systems for decades or even centuries.

Analytic modeling methods include fault tree models based on combinatorics and

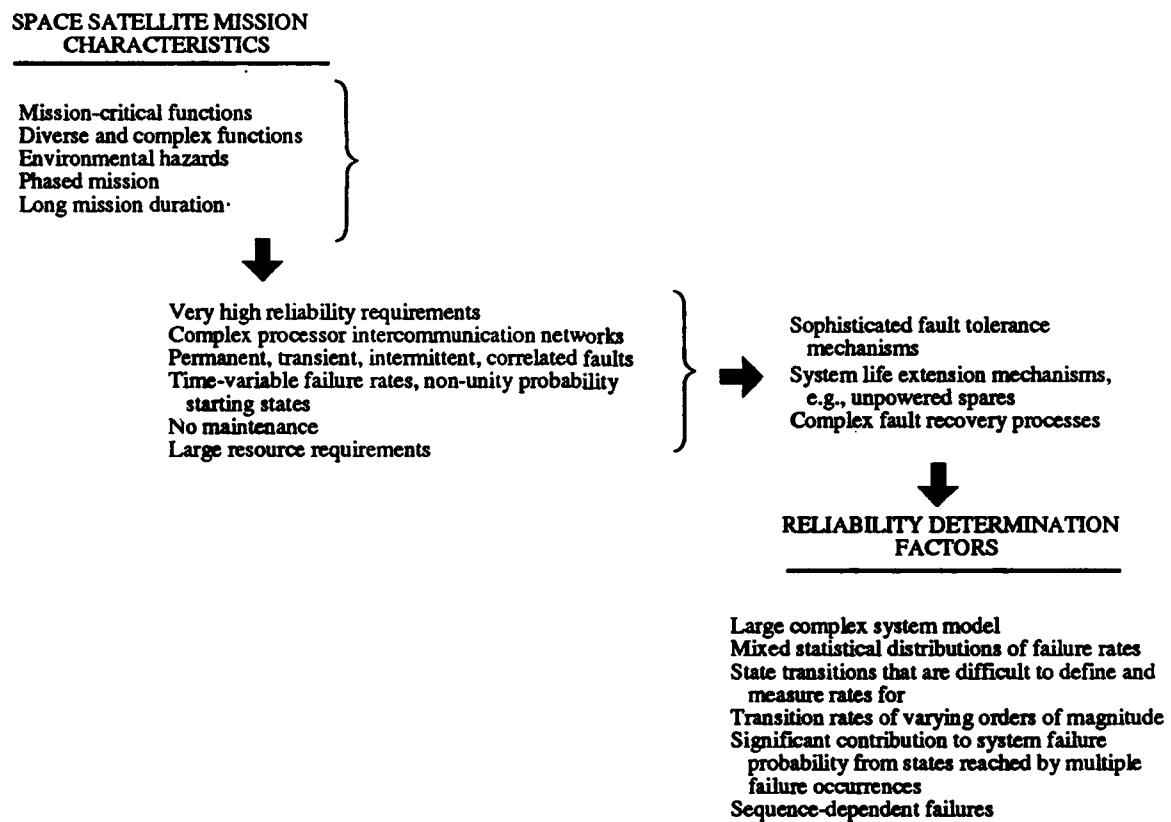


Figure 2.10. Impact of Mission Characteristics on Reliability Determination

Markov models based on discrete-state, continuous-parameter stochastic processes. The fault tree methods are generally applicable to systems that can meet their reliability requirements without reconfiguration mechanisms since the system state space can be represented as combinations of a finite number of elementary events. However, the systems of interest are at the ultra-high ranges of reliability requirements and must have sophisticated fault-recovery mechanisms. They are therefore likely to utilize dynamic reconfiguration strategies. Although the system state space is still finite, the probability of the system being in any state is a function of mission time and the rates at which fault arrivals and recovery processes cause transitions from one state to another. If the behavior of the system is such that the probability of being in a particular state in the future is dependent only on the present state, and if the transitions follow an exponential distribution, the system can be modeled as a Markov chain. Also, if the dependence on at most the prior state holds, but the transitions are not exponentially distributed, the system can still be modeled as a semi-Markov chain. These basic analytic modeling methods also rely on methods for constructing, simplifying, and verifying the models.

The results of reliability determination through analytic models are extremely sensitive to the accuracy of the data used in the model, the validity of the modeling assumptions for the system being modeled, and the fault models considered. Credible reliability determinations based on analytic modeling also rely on the theoretical capabilities and robustness of the method and the validity and numerical stability of the chosen tool for solving the model. Since the credibility of the reliability determination is so dependent on the validity of the model and since a number of assumptions and approximations have to be made in fitting a system to a model, model validation is a critical component for the determination.

The ability to construct experiments that can measure system processes to the degree required by the models constrains the level of detail to which system behavior can be modeled. Experimentation procedures and methods are necessary to accurately measure the most critical parameters. These methods include physical testing and simulation. One of the most important of the experimental methods is the injection of faults to observe system behavior and measure detection, isolation, and recovery times. Fault injection can be done with a working prototype of the system or with a gate-level model of hardware components, but, as demonstrated by the DAHPHRS program, can also be done on a behavioral simulation model of the system that includes application and operating system effects.

2.2.4. Fault Tolerance Analysis

The high level of reliability required for SDIO BM/C³ applications and the complexity of the architectures required to support those applications mean that mechanisms must be designed so that faults can be tolerated by the system. The goal of assuring that appropriate fault tolerance mechanisms are implemented relies on the ability to evaluate the proposed mechanisms. This evaluation has to assess the fault assumptions that have been made for all of the mission phases and the derived fault classes (and their error characteristics) that the system is designed to tolerate. It has to evaluate the specified fault containment regions and the fault isolation techniques. All error detection, fault diagnosis, error recovery, and system reconfiguration processes and architectural elements, including those that check the fault detection and reconfiguration mechanisms, must be evaluated and their effectiveness and detection times assessed. Additionally, the synchronization processes for redundant components, the provisions for fault-tolerant power and clocking, and the degree of independence of the fault tolerance mechanisms from the operating system must be evaluated.

An important part of assuring that appropriate fault tolerance mechanisms are designed into a system is the use of models during the design process to evaluate the effectiveness of the proposed mechanisms. This includes reliability models that incorporate the effects of fault detection, fault isolation, and system reconfiguration and that predict whether or not the system can meet its reliability requirements; system performance models that measure the performance impact of the fault tolerance mechanisms; and system functional models that can be used to determine the effectiveness of the fault tolerance mechanisms. As illustrated in Figure 2.11, these methods rely on the integration of models of the application; the operating system; the fault detection, isolation, and recovery mechanisms (FDIR); and the system architecture to evaluate the performance and functionality of the system and to provide parameters for a model to evaluate the reliability of the system.

The fault tolerance of a system is ultimately reflected by basic quantitative measures such as reliability and availability. For all but the simplest of fault-tolerant systems, establishing reliability requires that a comprehensive set of qualitative and quantitative evaluations be conducted. On the quantitative side, realistic estimates of important reliability model parameters, such as fault detection coverage, the time required to recover from a fault after it has been detected, and failure rates, must be established. Qualitative evaluations are required to assure that assumptions that are implicit in the construction of the reliability model are in fact satisfied. Examples of implicit assumptions include the assumption that the fault tolerance mechanisms are free of design faults and the assumptions regarding fault containment boundaries, types of faults, and synchronization of redundant channels.

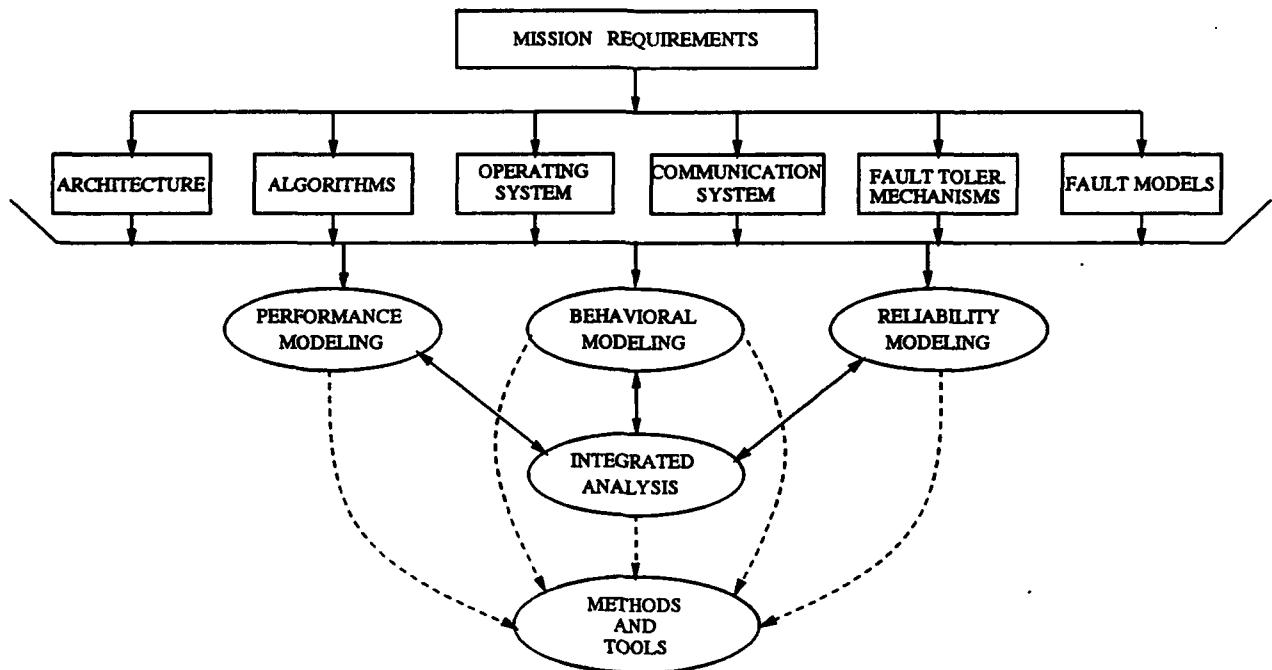


Figure 2.11. Models for Fault Tolerance Analysis

In the early- to mid-design phases, fault tolerance evaluation is of necessity incomplete due to lack of design detail. In the early phases, evaluation at best is restricted to qualitative assessments and parametric sensitivity analyses. The focus is likely to be on the design implications of high-level fault tolerance requirements and identified fault classes for establishing bounds for relevant parameters and for distinguishing between broad classes of architectural design decisions. As the design progresses, fault tolerance evaluation can be more quantitative and the qualitative analyses can be more thorough. The focus can shift to finer discrimination between competing designs, establishing viability of a particular design, identifying design deficiencies and areas needing improvement, determining more realistic estimates of parameters such as recovery time or detection coverage, determining the behavior of fault tolerance mechanisms in the presence of faults, and finding and eliminating concept- or requirement-level design errors in the fault tolerance mechanisms.

Also in the early- to mid-design phases, models of appropriate elements of the system design can be constructed and used to support some of the qualitative and quantitative evaluations described above. Neither performance modeling, which focuses on workloads, nor reliability modeling, which focuses on transitions from operational to failed states, are sufficient to evaluate system fault tolerance mechanisms. Evaluation of fault tolerance mechanisms requires a more detailed representation of system behavior. In addition to workload and operational and failed states, the functionality

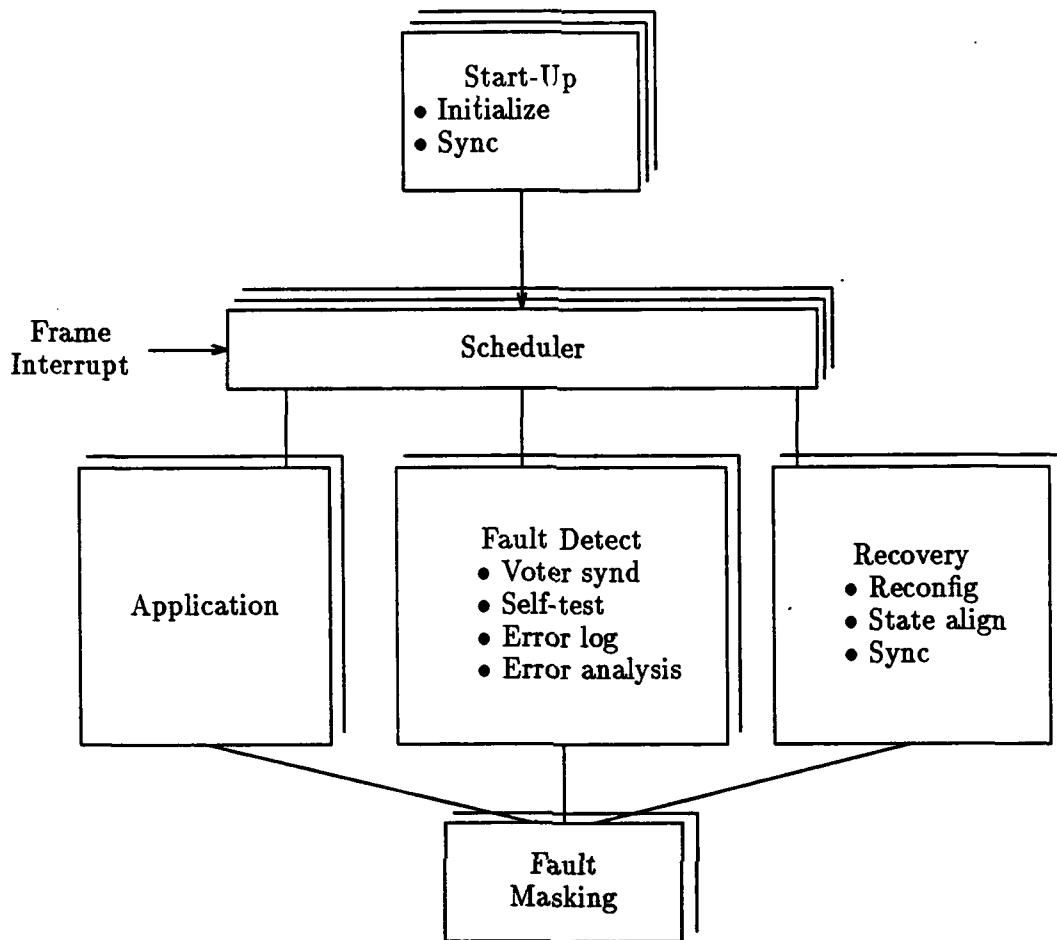


Figure 2.12. Integrated Model

of each system element must be represented under normal and faulted conditions. Specific data and control variable values, as well as data and control flow, become important. The effects of all system elements such as the application software, the hardware architecture, the software operating system, and the fault detection, masking, and recovery mechanisms must be represented. Elements of the system design whose behavior must be captured by the models include the FDIR mechanisms, which are likely to be realized by a combination of hardware and software, the hardware architecture, the software operating system components, such as the task scheduling function, and the application software that implements the functional requirements of the system. While each of these models can be evaluated separately, a functional/behavioral model that integrates these models, such as that illustrated in Figure 2.12, is required to capture the interactions between the FDIR mechanisms and the application and operating system functions.

The component models of the integrated model have to capture the performance characteristics and the functionality of the system elements, and the integrated model has to capture the behavior of the total system in operation. This behavioral model can then be used to assess the performance and correct operation of target system elements under faulted and fault-free conditions. The ability to inject faults and observe the resulting system behavior is provided by the capability of the behavioral model to simulate the actual processing of data at a functional level. This allows the performance and effectiveness of FDIR mechanisms to be assessed and provides estimates of reliability model parameters that are usually not known until measurements can be made on prototypes.

2.2.5. Integrated Performance and Reliability Analysis

More accurate predictions of a fault-tolerant system's performance and reliability early in the design cycle facilitate the architectural and algorithmic trade-offs necessary to ensure the implementation of a system that meets the application requirements. The goal of integrated performance and reliability analysis is to enable the assessments of a design and to facilitate the transfer of data from one type of analysis to the other so that through an iterative design process a reasonable trade-off between performance and reliability can be attained. The key to achieving this goal is the ability to include the fault tolerance mechanisms in the performance analysis as well as in the reliability analysis and to correlate the results from each type of analysis with the assumptions of the other.

The DAHPHRS methods for achieving this goal are based on the assessment and comparison of system performance, reliability, and fault tolerance through hierarchical models of application algorithms, architectures, operating systems, and fault tolerance mechanisms, as described in the preceding sections. Figure 2.13 illustrates the relationships between the various models and the integration of performance models into a system performance/fault tolerance model.

The system performance/fault tolerance model requires two distinct types of performance models, depending upon the level of abstraction appropriate for the design phase. The integrated functional/behavioral model described for fault tolerance analysis can be used to assess the operation of targeted system elements under faulted and fault-free conditions and to assess the effectiveness of fault isolation, error detection, fault diagnosis, error recovery, system recovery and sparing strategies. The performance attribute model described for performance analysis can be used in the early, high-level analyses or in conjunction with the functional/behavioral model in later, more-detailed analyses to determine response times, resource utilization of the various fault tolerance strategies, and the system capacity in various degraded but

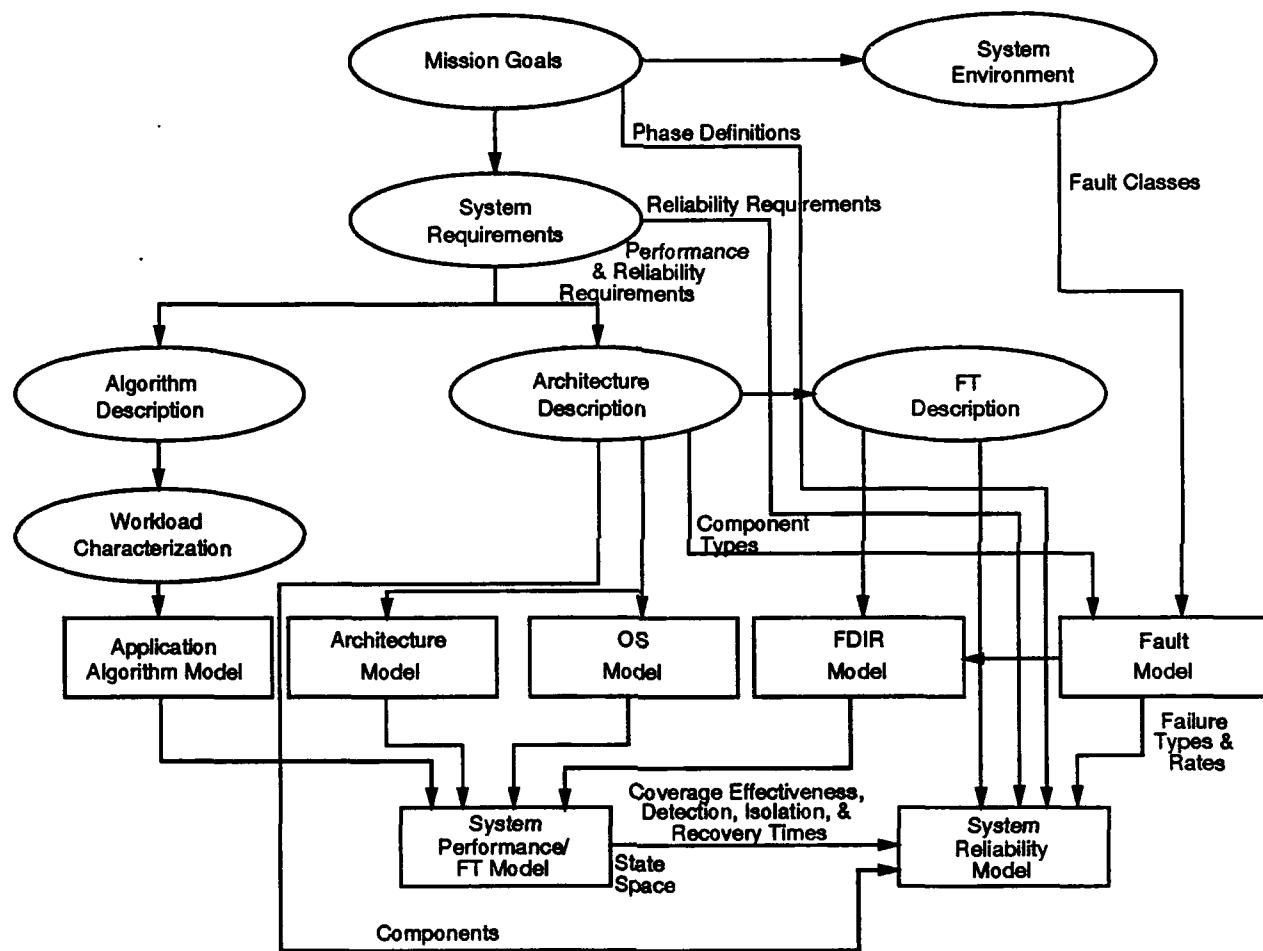


Figure 2.13. Models for Integrated Performance and Reliability Analysis

operational states. Reliability models, using results from the analyses of the system performance/fault tolerance model, can be used to determine the reliability of various design configurations and can be used to determine required architectural parameters such as sparing levels, redundancy levels, or maximum allowable reconfiguration time.

The use of these models in each of the three phases enables performance/reliability trade-offs such as 1) the number of processors required for reliability vs. computation, 2) the additional reliability gain from increased coverage of faults vs. the performance cost of providing that coverage, and 3) how the functionality of the FDIR mechanisms impinges on the design and functioning of the application algorithms.

2.3. Required Tools

The model-based design and assessment methods require tools to create the descriptions and models specified in Figure 2.13, to ensure consistency between the different models, and to facilitate the transfer of accurate data among the models. The methods also require simulators for the attribute, functional, and behavioral models, static analysis tools for the attribute models, and reliability analysis tools. Since measured data from prototypes play an essential role in refining and validating models, tools for creating and instrumenting prototypes are also needed. Figure 2.14 illustrates how these tools would interact in an integrated analysis environment. The data base in this figure would include the system descriptions and the analysis results. Together with the experiment manager, it would aid in ensuring consistent models and in transferring data among the models.

The model creation, simulation, and analysis performed during Phase I and Phase II of DAHPHRS relied on existing tools. These tools are summarized in Table 2.1.

2.3.1. Application/Algorithm, Architecture Description, and Workload Characterization

The information required to build models includes the mission and application description, system requirements, algorithm descriptions, architecture description, and a workload characterization. The algorithm and architecture descriptions are built up as the system is designed from the application/mission description and system requirements. The necessary information that must be derived from the architecture description is summed up in Tables 2.2 and 2.3.

The primary concern of the algorithm description is to identify the type and degree of

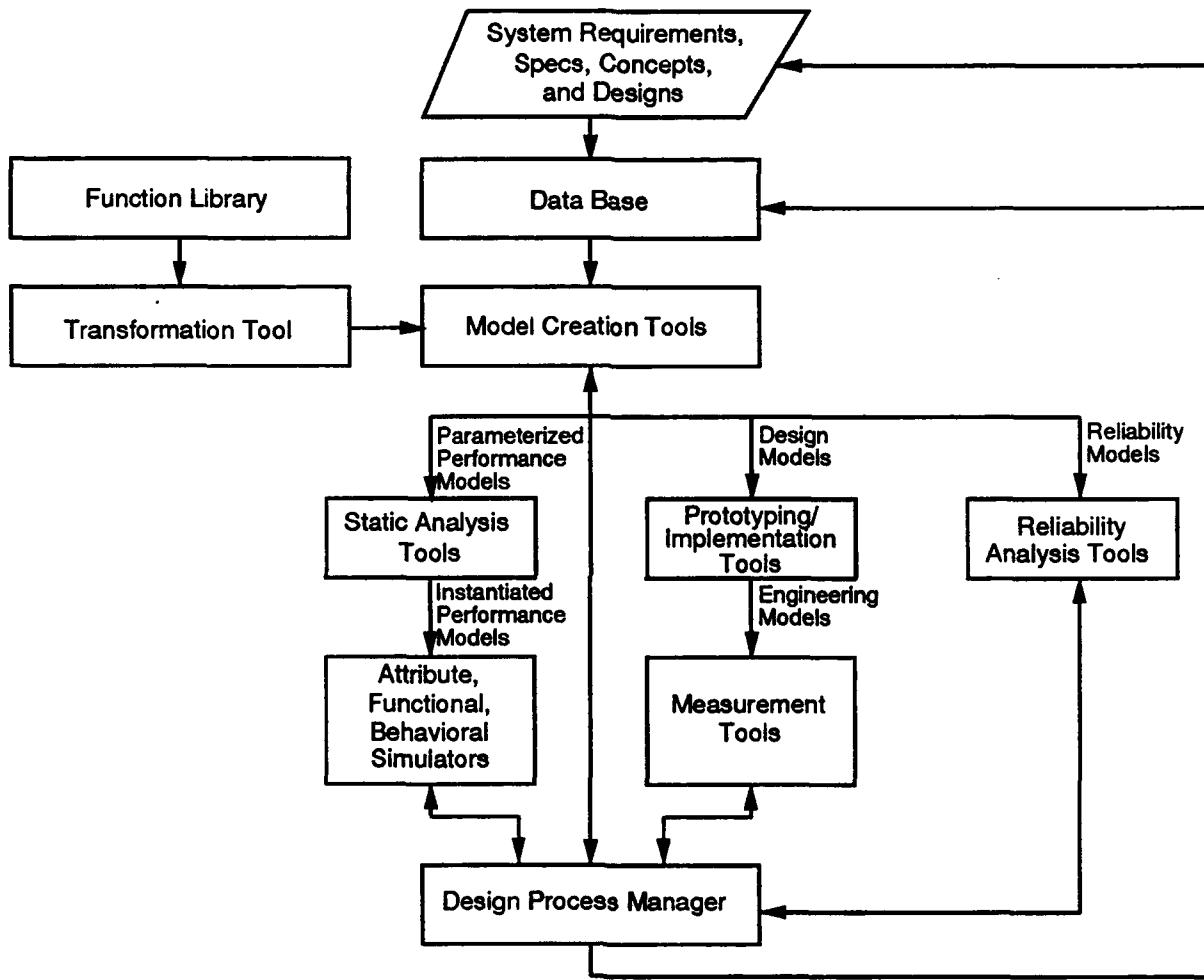


Figure 2.14. Integrated Tools

Table 2.1. Existing Tools

Tool Used	Type	Purpose
Teamwork	CASE	Algorithm description
ADAS/EDIGRAF	Attribute model creation	Parameterized, hierarchical model creation
ADAS/ADLEVAL	Parameterized model definition	Algorithm description, function library
ADAS/ADLEVAL	Static analysis	Instantiated models
ADAS/GIPSIM	Performance modeling via data flow and resource allocation	Performance estimates
ADAS/CSIM	Functional modeling and simulation	Examine functionality of design
ADAS/CSIM	Stochastic modeling	Expand classes of systems that can be modeled
ADAS/CSIM, Ada	Behavioral modeling and simulation	Compare functionality, integrate effects of subsystems, represent fault effects
Instrumented Code, Prototype Testbed	Measurement	Provide performance data for refining performance models
ASSIST/SURE	Parameterized attribute model creation/ reliability modeling	Probability of system failure with respect to mission time

Table 2.2. Architecture Description Information Requirements

System Hardware Information	
Components	Required Information
power modules processor modules memory modules timing/clocking modules fault-handling hardware modules processor cache data exchange hardware	functional specification interfaces timing requirements estimate of design complexity & and logical structure fault classes and failure rates processing speed memory read/write times capacities volatile control registers & memory software-resettable hardware clock-deterministic hardware phase-locked oscillators redundancy level number of spares
communication links & modules: I/O, data acquisition and distribution, interprocessor data buses, internal processor bus	topology and interconnections protocols performance message structure path allocation timing, latency redundant paths transfer rates capacity bandwidth communications overhead
fault models	type frequency mission phase or technology dependence error characterization: (count, origin, activity, duration, extent, temporal behavior and cause of multiples)
fault detection, isolation, and recovery	local & global error detection, error masking & fault containment mechanisms local & global redundancy management, recovery & reconfiguration strategies compensation strategies for delays and/or lost information data consistency protocols sparing strategy function migration/reassignment protection of critical functions synchronization & start-up self-test state alignment partitioning into hardware & software detection of faults in FT mechanisms exception handling times and effectiveness measures

Table 2.3. Architecture Description Information Requirements

System Software Information	
Components	Required Information
task scheduling	description of control characteristics (distributed, central, or hierarchical)
I/O services	function overheads
interrupt services	expected execution times per function
memory management	function response times
utilities	uncertainty in response times
interprocessor communications services	software semaphores
RAM alignment software	reserved addresses in memory
initialization software	criticality of function
synchronization software	
functions related to fault tolerance	

detail of the information that is required to ensure adequate fidelity of the performance modeling at the various stages in the design process. It is also important to assure that in the early stages of the design the algorithm descriptions are not so detailed as to preclude potentially desirable system configurations. In Phase I, descriptions were created for two mission planning algorithms: Weapon-to-Target Assignment/Target Sequencing (WTA/TS) and WAUCTION_ASSIGNMENT [3]. These descriptions are described in [5], [6], and [2].

Considerable effort is required to develop the algorithm descriptions used for the development of performance models. The description should include a diagram of the major subfunctions within the algorithm. For each subfunction, the inputs, processing, and outputs must be described. Where appropriate, subfunctions should be further decomposed and inputs, processing, and outputs identified and described for each of those components. Data inputs and outputs should be characterized by source or destination, by quantity as functions of system parameters such as numbers of targets or weapons, and by type such as numerical or logic variables. Subfunction processing steps should be described wherever possible. To the extent possible, data dependencies, data addressing patterns such as linear or random, and opportunities for parallel decomposition should be included in the descriptions. Diagrams, such as N-square diagrams, indicating all inputs and outputs for each subfunction and their relationship to other subfunctions, should be included in the descriptions. Finally, real-time processing constraints should be specified. This information is summed up in Table 2.4.

The effort required to develop the algorithm descriptions could be reduced by including additional features in the modeling tools. Use of computer-aided software engineering (CASE) tools to specify and create the algorithm descriptions would be a more effective way to handle the development of algorithm descriptions. One such tool is *teamwork*, by Cadre Technologies, Inc.[7]. It provides the user with a graphical

Table 2.4. Application/Algorithm Description Information Requirements

Application/Algorithm Information	
Components	Required Information
inputs/outputs	characteristics structures ranges quantity rates
functional decomposition	description inputs/outputs data flow between subfunctions time constraints
subfunctions	inputs/outputs time constraints processing or computations transport delay, process update rate
partitioning to parallel processors	minimum granularity dynamic load balancing requirements
distributed functions	control, scheduling, synchronization, communications data-flow timing, throughput, bandwidth, latency, and memory locality and interaction of references
critical functions, fault tolerance interactions	maximum computing delays critical state information mechanisms for improving robustness in the presence of faults maximum allowable performance degradation

interface useful in creating process bubbles, data stores, data and control flows, and many of the other building blocks for structured analysis and design described by Hatley/Pirbhai in [8]. The *teamwork* environment also provides a hierarchical system for building multiple levels of abstraction until a terminal process specification or control specification is required.

A workload characterization, or computational model, is required to build models of application algorithms that can be integrated with models of system architectures. This characterization should provide workload by function in terms of machine instructions, include rules for decomposition, identify parallelizable components and sequential components, and include processing workload and communications workload. It is constructed by deriving the following items from the algorithm description and the application/mission requirements:

- data dependencies and addressing patterns
- data-flow timing
- opportunities for parallel decomposition
- real-time processing constraints, data rates, transport delays, and process update rates
- estimated operation counts
- memory access counts
- control, scheduling, synchronization, and communications
- throughput, memory, and bandwidth requirements
- locality and interaction of references
- maximum allowable performance degradation

2.3.2. Performance Modeling Tools

Performance modeling tools must support the analysis of multiple models of varying levels of detail and complexity as described in Sections 2.2.2, 2.2.4, and 2.2.5. Therefore, they must be able to create parameterized, hierarchical models. Hierarchical models allow different levels of abstraction to be represented and analyzed. This allows high-level models created in the earliest design stages to be developed in more detail as the design progresses. It also allows the system engineer to select the level of

detail appropriate for components of the model for particular analyses. Hierarchical models also need to support the synthesis of data at higher levels in the model from analysis of lower level components and the inheritance of data from the higher levels to the lower levels. The ability to select among the various levels of a model is especially useful in integrating several models for performance or fault tolerance analysis since it is unlikely that the lowest level of detail of each of the component models can be captured in the integrated model.

Parameterized attributes allow ranges of analyses to be conducted over varying system and mission characteristics and architectural structures and make different representations of model attributes possible for different model levels. The necessary types of representation are fixed, variable, stochastic, and dynamic. With these different representations of attributes, descriptions of aspects of system behavior can be captured as appropriate for the particular level of modeling required by the level of design or by characteristics of the system. Parameterized attributes also enable the synthesis and inheritance of data throughout the model hierarchy and make the creation of generic, reusable models feasible.

Generic software models are particularly important in the evaluation of parallel systems since a generic model of an algorithm can be created, then instantiated and transformed to reflect its implementation on a particular architecture. Thus, all models of that algorithm for various architectures originate from the same core model. Generic models of frequently used functions can also be built, validated, and used as components of models of application algorithms that use them. The generic models can be stored in a function library.

The least detailed level of modeling requires tools that can perform a static analysis of performance characteristics based on specified attribute values of the models. At the next level, performance modeling tools must support the simulation of algorithmic processes mapped, or constrained, to a structural model of the architecture through the use of models that capture data and control flow. As more detailed analysis is required, the tools must be able to simulate functional models of either the algorithms or the architectural components. Finally, they must be able to incorporate various levels of system component models into a system behavioral model that captures data flow, control flow, functionality, and changes in data value; that models the effects of operating system and scheduling functions, architectural components, fault tolerance mechanisms, and application algorithms; and that can represent fault effects. Throughout the various levels of modeling, the tools and models must be able to incorporate data from engineering and other measurement models so that model parameters can be derived from measurements and statistical data.

The Phase I and Phase II performance modeling was done using the Architecture Design and Assessment System (ADAS). ADAS is a tool developed by RTI for the hi-

erarchical description and assessment of system designs. In ADAS, the system performance model is created from a structural model of the architecture and a data/control flow model of the processes. The structural model, or ADAS hardware graph, is a directed graph comprising nodes to describe the architectural components and arcs to describe the connectivity among components. Attached to each node and arc of the graph are attributes that become constraints in the performance model based on the construction of a mapping from the structural model to the algorithm model. The algorithm model, or ADAS software graph, is a directed graph describing the data/control flows (arcs) of the software processes (nodes) in the system. Its attributes define the required computing and communication resources and control the assignment of software components to hardware components. The constrained software graph created by the construction of a mapping and containing processing times defined by the attributes of both graphs becomes the ADAS performance model. This performance model can be simulated by the ADAS tool GIPSIM to predict the performance of the software processes on the architecture for that mapping and thereby predict whether the system will attain its throughput requirements. In ADAS, a CSIM functional simulation model can also be constructed to simulate the function of software processes, such as application or FDIR algorithms, or the detailed operations and interactions of the system components.

In recent modifications to ADAS, an Attribute Definition Language (ADL) and ADL Evaluator (ADLEVAL) have been developed for the creation of parameterized performance models. ADL expressions describe the performance model attributes in terms of system parameters. These expressions are associated with ADAS nodes, arcs, and graphs, and are translated into ADAS attributes by ADLEVAL. Also, the ADL expressions can be inherited and synthesized throughout the hierarchy of graphs that describe the system so that, for example, processor instruction processing speeds, memory and interconnect bandwidths, and mission parameters can be included in an ADL file at the root graph and be accessed at all subsequent graph levels.

The evaluation of fault tolerance performance for a parallel architecture for the design refinement modeling phase requires the capability to capture data and control flow, to simulate the actual processing of data at a functional level, to inject and react to faults, and to integrate a large number of submodels into a behavioral model. This capability does not exist in current tools to a level sufficient to make the creation and simulation of such a model feasible. To demonstrate the needed modeling capability and to reduce the complexity of allowing flexibility in including different models of the various processes for different simulation experiments, an object-based discrete-event simulator was created. The simulator was written in Ada to take advantage of the Ada tasking facilities. Each system component was modeled by an Ada task and their interactions were modeled by task rendezvous under the control of a system schedule task. One of the benefits of using an Ada-based modeling technique is that the algorithms can be expressed in the same manner in the model as they will be

implemented in the target architecture, allowing the model development effort to become a part of the code implementation effort.

All performance models have to be derived from system and requirements descriptions, and tools, such as CASE tools, are needed to construct these descriptions. Tools are also needed to facilitate the construction of the performance models from these descriptions. This includes tools for generating workload characterization as well as the structural and data- and control-flow components of the model. The models in turn could be transferred to the performance simulation tool via an appropriate tool interface. Model construction effort could be further reduced by incorporating a library of models for a wide range of common processing algorithms.

2.3.3. Reliability Modeling Tools

Reliability modeling tools must support the reliability analysis process discussed in Section 2.2.3. Once a particular method or set of methods has been determined to be applicable for evaluating the reliability of the system under consideration, it is necessary to carefully consider which of the existing tools that implement that method is best suited to the system characteristics. Although a number of tools have been developed based on Markov models, most contain simplifying assumptions that limit the class of systems to which they can be validly applied.

These tools must be able to determine such components of reliability as the number of spares required, the redundancy level required, the sensitivity of system reliability to the performance and effectiveness of proposed fault tolerance mechanisms for the expected class of faults and the number of processors available at given mission times, as well as the probability of system failure with respect to mission time. These tools have to 1) handle phased missions, ultrareliable systems, and very large and complex systems; 2) analyze complex processor intercommunication networks; 3) handle multiple types of faults; 4) accommodate time-variable failure rates; 5) allow system starting states other than zero failures with unity probability; 6) model complex fault recovery processes; 7) allow for cool spares with reduced failure rates until activated; 8) handle time sequence dependencies between certain faults; and 9) allow for multiple near-coincident faults.

Reliability modeling relies on tools to construct parameterized attribute models that can be refined to capture increasingly detailed descriptions of system behavior in the presence of faults and that can be used to conduct sensitivity analyses for a range of assumptions. It also relies, as does performance modeling, on measurements from prototypes for refined estimates of model attributes. Given the reliance on state descriptions for reliability estimation and the large state spaces that result from

complex fault-tolerant architectures, reliability modeling tools have to have model reduction mechanisms for both model creation and solution.

Markov (including semi-Markov) modeling provides a flexible way of describing fault-tolerant systems, and tools such as ARIES[9], CARE III [10], HARP [11], and SURE[12] have been developed to compute solutions of the models. However, since the Markov state descriptions increase rapidly with system size and complexity and the solutions are computationally difficult, tools that are based on Markov models have inherent limitations. Additionally, some tools have made limiting assumptions regarding fault handling behavior to reduce the computational complexity. In general, no current reliability analysis tool can address all of the complex interactions of highly reliable, high-performance systems. Even if the underlying assumptions are flexible enough to describe the system, the complete system model would be too large to be solved. Furthermore, no tools exist to assess the reliability of the complex multiprocessor networks found in these systems.

The reliability modeling in Phase I and Phase II was performed using reliability estimation tools provided by NASA Langley Research Center, including ASSIST [13], SURE, and STEM[14]. SURE computes an upper and lower bound for the probability of entering a death state of a semi-Markov model containing recovery transitions with arbitrary statistical distributions. It is fast and accurate for these models because it computes the bounds based on algebraic properties of the means and variances of recovery times rather than directly solving the differential/integral equations of the model. More important for this application, there are no underlying assumptions regarding fault handling behavior that preclude modeling advanced fault-tolerant systems such as the FTPP. Additionally, the ability to define recovery transitions by the mean and variance of their distribution facilitates the use of experimental data for those transitions in the model.

STEM computes death state probabilities of Markov models based on the assumption that all distributions are exponential. It was used in modeling cases where long mission times and/or large numbers of system components resulted in long compute times for SURE's path generation and analysis.

ASSIST was used to create models in the input form required by SURE and STEM. Independent of any tool's ability to compute a solution for a particular model, the creation and validation of a correct reliability model is difficult. Thus it is important to have a tool such as ASSIST that can create a parameterized model from a description of the system's fault occurrence and handling behavior. Such a tool not only facilitates the creation and subsequent modification of large models, but also provides documentation for the model and a means of communicating a system's fault tolerance description among different project personnel and across different phases of the development process.

2.3.4. Summary

Tools can provide valuable insights into design decisions at the early stages in the design process; however, the current set of tools is unable to handle the size or complexity of SDI algorithms and architectures effectively. There do exist methods for reducing the complexity of the models to the point where they can be solved by existing tools, but the construction and validation of the reduced models is not well understood. Further, the current tools were built to address performance and reliability analysis separately. Thus, it is difficult to ensure consistency between the different models and to transfer data among the models.

A preliminary environment for the integration of tools based on common model generation, experiment management, and data base tools is illustrated in Figure 2.14. The specification of the requirements for the tools and their integration into this environment which are scheduled for Phase III will derive in part from the following needs identified in Phase I and II:

- The development of requirements via CASE tools and mechanisms to create performance models from the CASE descriptions are needed.
- Mechanisms for bringing fault tolerance mechanisms into performance models are needed.
- The effects of the software operating system need to be brought into the modeling process.
- Methods for integrating functional models of system components into a system behavioral model are needed.
- The systematic generation of reliability models and tools for model reduction are needed.
- Tools for network reliability analysis are needed.
- Measurement is necessary to support and validate modeling.
- Tools to support experiment management are needed.
- A shared data base consisting of the following elements is required:
 - the data required for performance and reliability models
 - a library of primitive function models
 - architecture transformation rules
 - parallelization transformation rules

– mapping rules

- Tool support for validating models is needed.
- A proper match of model resolution to tool capabilities must be maintained.
- Sophisticated mapping and scheduling tools are needed.
- Tools need to be sensitive to parameter changes in the algorithm.
- Run-time modification of hardware attributes to model dynamic resource management is required.

3. Baseline Determination Modeling Phase

3.1. Description

The baseline determination modeling phase corresponds to the first iteration through the design steps illustrated in Figure 3.1, where the identification of fault sets, the

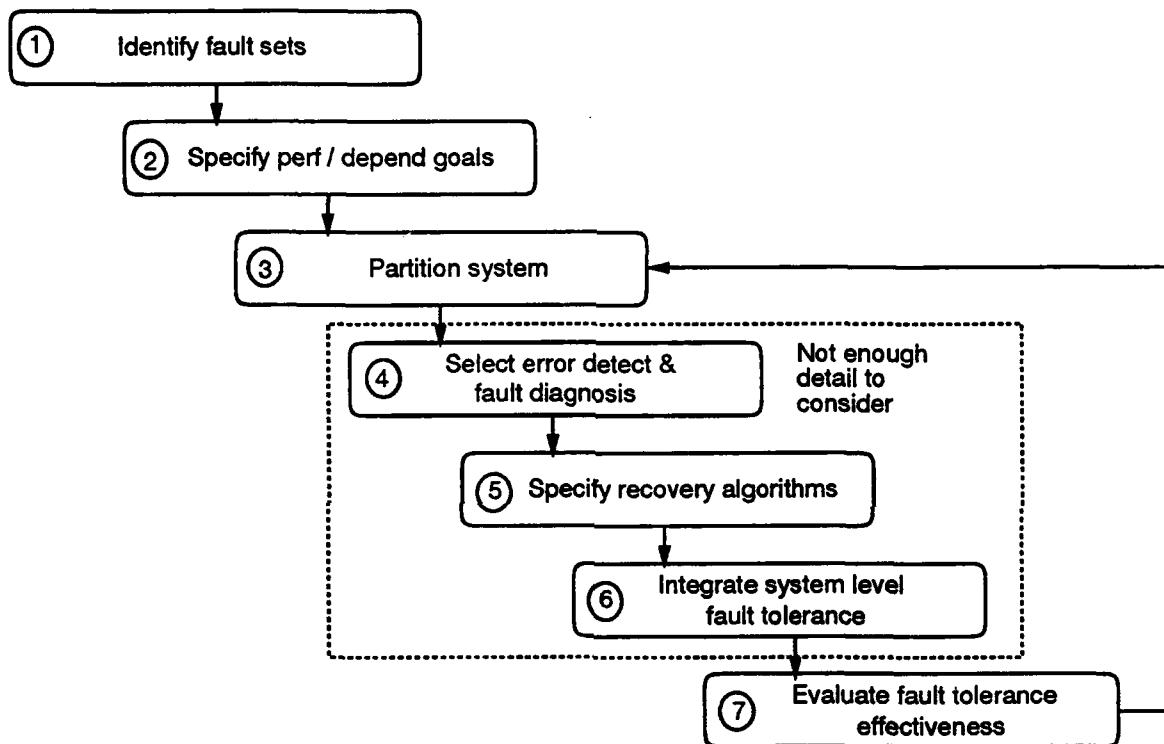


Figure 3.1. Baseline Determination Relationship to Methodology

specification of performance and dependability goals, and the system partitioning are of prime importance. It is during this stage of design that high-level design alternatives are developed by defining the basic architectural and algorithmic structures and characteristics and evaluating their potential to meet the requirements of the application and mission. As indicated in Figure 3.1, specific fault detection, isolation, and error recovery strategies have not been developed. In this early phase, evaluation is restricted to quantitative assessments and parametric sensitivity analyses. However, it is crucial to begin these high-level analyses at the very first iteration to guide the design process. The focus is on the design implications of high-level fault tolerance requirements and identified fault classes for establishing bounds for relevant parameters and for distinguishing between broad classes of architectural design decisions.

Figure 3.2 depicts the model development and analysis for this phase and indicates the type of information to be transferred between the performance modeling and the reliability modeling. The performance models utilized at this level are the parameterized attribute models of Figure 2.8. These models need to support simulation based on data flow and a mapping of application/algorithm functions to architectural components. The reliability analysis requirements need to be developed at this stage according to the process illustrated in Figure 2.9. At this level of design, a high-level parameterized model is needed to support the relevant analyses. The types of tools necessary to support the baseline determination phase of modeling are listed in Table 3.1.

Table 3.1. Tools for Baseline Determination Phase

Tool Used	Type	Purpose
Teamwork	CASE	Algorithm description
ADAS/EDIGRAF	Attribute model creation	Parameterized, hierarchical model creation
ADAS/ADLEVAL	Parameterized model definition	Algorithm description, function library
ADAS/ADLEVAL	Static analysis	Instantiated models
ADAS/GIPSIM	Performance modeling via data flow and resource allocation	Performance estimates
ASSIST/SURE	Parameterized attribute model creation/reliability modeling	Probability of system failure with respect to mission time

As indicated in Figure 3.2, the required information for this level of modeling consists of application and mission requirements. From this information, a coarse workload characterization can be made and combined with a high-level architecture description to create a performance model. Relative to the fidelity of the workload characterization of the application algorithm in terms of machine instructions, the following performance measures can be made using the models and tools at this level of design:

- upper and lower bounds on workload
- utilization of resources by application function
- effect of contention for resources on application compute time and on utilization of the resources

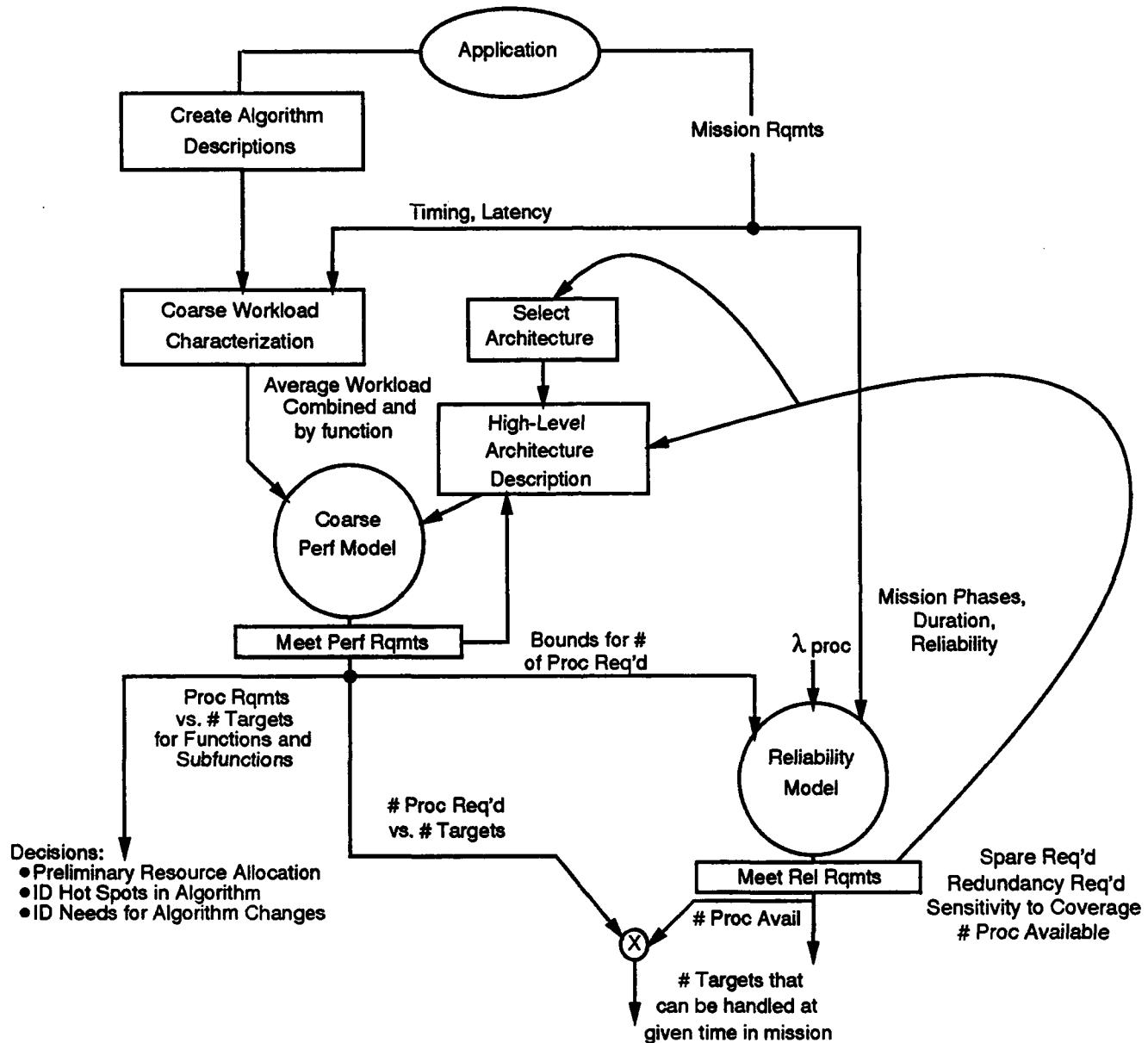


Figure 3.2. Baseline Determination Phase Process Diagram

From analyses of the performance model, bounds on processing resources required to meet the performance requirements can be determined in terms of mission parameters, such as number of processors required versus number of targets to be handled. Also, decisions can be made for preliminary resource allocations, and potential problem areas or areas requiring special attention in application algorithms can be identified.

Based on the architecture description, the mission and application requirements, and the results of a performance analysis to determine bounds on required processing resources, a high-level reliability model can be created. From this high-level reliability model, bounds on the probability of system failure with respect to mission time and on the expected value of the number of operational components at a given time can be estimated. Although no information is available about the functioning of the fault detection, isolation, and recovery procedures, an estimate of the sensitivity of system reliability to the handling of faults can be made through parametric analyses of a model that incorporates a probability of fault handling success, or coverage.

From analysis of this model, the redundancy and sparing levels necessary to maintain adequate processing resources in the presence of faults throughout the duration of the mission can be determined. Results from the reliability analyses can also be used to identify changes to the architecture that may be required to meet the reliability requirements. From these analyses the available processing resources at particular points during the mission can be determined and, as for the performance analyses, can be expressed in terms of mission parameters.

Given the performance analysis estimates of the number of processors required to handle the workload and the reliability analysis estimates of the number required to achieve the specified level of reliability, trade-offs between resources required for reliability versus performance can be made.

At the end of the baseline determination modeling phase, the system engineer will have high-level architectural designs and preliminary resource allocations for those designs. He will also have high-level models of the system that can be expanded hierarchically to a lower level of detail as the design progresses. The model parameters can also be refined as more detailed information and more accurate measures become available for the activities that they represent.

3.2. Baseline Determination Case Studies

During Phase I of the DAHPHRS program, a number of performance modeling case studies were conducted to explore issues of model construction, fidelity, and use at this level. These case studies illustrated the use of models to evaluate the relative

effectiveness of various granularity levels and algorithm decompositions and to aid the analysis of interprocessor communications, of data/program dependencies, and of resource utilization and memory sizing. Of particular interest was the use of models to investigate the impact of network communications on performance, the impact of fault tolerance on communications, the speedup and effectiveness achievable through varying levels of parallelism as a function of both processor/communication speed and the existence of inherently sequential components of tasks, the effectiveness of various embeddings of algorithms in architectures, and the distribution of workload by function and by processing resource.

High-level reliability modeling case studies were conducted in Phase I to investigate issues in modeling large systems at the baseline determination level. High-level analyses were conducted that are appropriate to this phase and that can be used to determine sparing levels required to keep a system operational for a long mission, to determine the impact of operating at degraded performance levels on extending system life, and to estimate network reliability.

The Phase I baseline determination case studies are summarized in Table 3.2 and documented in [2].

Table 3.2. Summary of Phase I Case Studies for Baseline Determination Phase

Modeling Area	Analysis Area	Algorithm	Architecture
Model Development	High-level workload and workload distribution	WTA/TS	
		WAUCTION	
	Architectural structure		Hypercube
			Multimax
			FTPP
Performance Simulation	High-level decomposition	Generic	Generic
	Parametric studies	WTA/TS	FTPP
	High-level interprocessor communications	WTA/TS	FTPP
Reliability Analysis	Spares	Generic	
	Degraded performance	Generic	
	Network reliability		

It was found in Phase I that although high-level models were useful, the ability to quickly construct performance models at the right level of abstraction and for varying degrees of parallel decomposition was essential for investigating multiple architectural and algorithmic decomposition alternatives. Therefore investigation of the creation and use of a library of primitive functions with transformation rules was undertaken

in Phase II. Results from Phase I models indicated that since speedup is relative to the processing-workload-to-communication-workload ratio inherent in the function, methods of static analysis of algorithm and architecture granularity ratios should be investigated in Phase II. Finally, the Phase I network reliability case study prompted the conduct of a survey of analytical network reliability estimation methods.

3.2.1. Function Library

3.2.1.1. Background

In systems design, modeling is an important process. System models often contain common functions or subfunctions that need not be re-created for each new system design. The existence of a function library containing these common functions or modules can greatly reduce the amount of work required to model complex systems. This task was undertaken to demonstrate the creation and use of such a function library.

The Weapon-to-Target Assignment/Target Sequencing (WTA/TS) algorithm from Phase I of this effort was selected for demonstrating a function library. The description of this algorithm created during Phase I was refined in Phase II to identify matrix operations, sorting operations, and linear and integer programming operations utilized by the algorithm. Based on this analysis, functions were selected for the library that could be incorporated in the WTA/TS algorithm model for these operations. An example library was then constructed and verified. The use of the library was demonstrated in a comparison of the performance of Phase I and Phase II versions of the WTA/TS algorithm.

3.2.1.2. Description

Figure 3.3 contains a diagram of the methodology for creating and using the function library. The systems design process shown here begins with two Architecture Design and Assessment System (ADAS) graphs, an algorithm graph and its corresponding target architecture graph. Nodes in a graph may have subgraphs that are functional decompositions of themselves. Each graph or subgraph element throughout the hierarchy has a corresponding Attribute Definition Language (ADL) file. The function library consists of ADAS graphs of functions and their corresponding ADL files. The algorithm graphs and subgraphs and their ADL files are inputs to a static analyzer, shown here as ADLEVAL, which evaluates designer-specified equations to estimate the system's expected operating parameters. The results of this analysis are then used to dynamically model the system using GIPSIM. From the combined use of ADAS and the function library, one can readily determine design parameters such as overall and functional workload, resource utilization balancing, and communication and fault tolerance overhead.

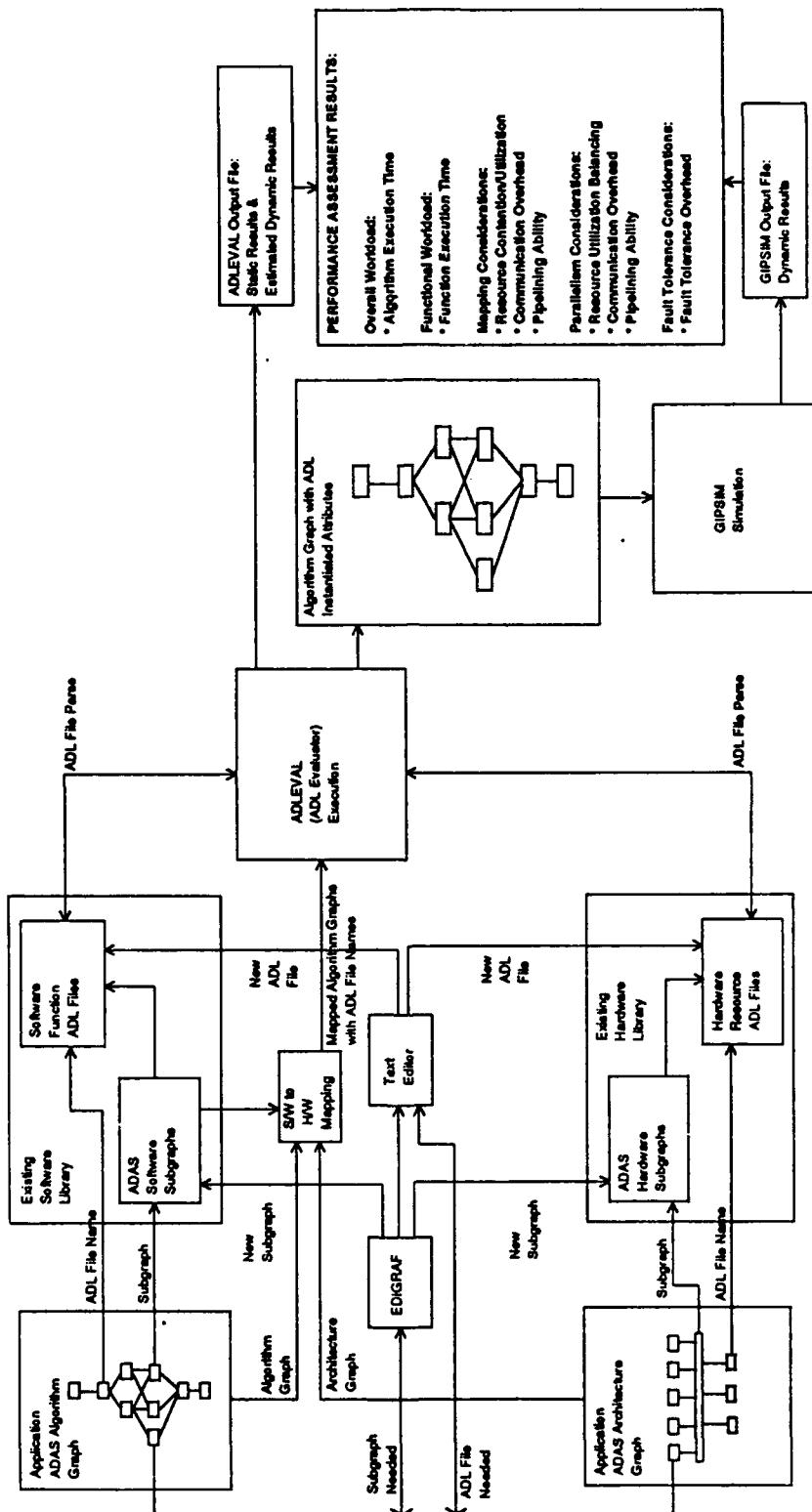


Figure 3.3. Methodology for Use of Function Library

3.2.1.2.1. Library Creation

As in Phase I, the revised Phase II algorithm description was created by decomposing the WTA/TS algorithm into its constituent functions. At the highest level, these functions are target cluster definition (TCD), weapon-to-target cluster allocation (WTC), and weapon-to-target assignment (WA). These functions were, in turn, decomposed into their subfunctions, down to the level of identifiable matrix, sorting, linear programming, and integer programming operations. From this decomposition, the following common functions were identified for inclusion in the function library:

- `Column_Sum_Module.adl` — “Sum over Columns of a Matrix” function. This is the first component of the Cluster/Assignment Formation function.
- `Diag_Elements_Module.adl` — “Set Matrix Diagonal Elements” function. This is the third component of the Cluster/Assignment Formation function.
- `Fill_Matrix_Module.adl` — “Fill Matrix” function. This is the fourth component of the Cluster/Assignment Formation function.
- `G_L_M_Module.adl` — A function referred to as the GLM or Dual Problem function, consisting of the following three components:
 - `Gradient_Module.adl` — “Gradient” function
 - `LaGrange_Update_Module.adl` — “LaGrange Update” function
 - `Median_Measure_Module.adl` — “Median Measure” function
- `NlogN_Sort_Module.adl` — “NlogN Sort over Vector Elements” function. This is the second component of the Cluster/Assignment Formation function.

Based on the revised decomposition and identification of operations, the data variables used in the algorithm were identified. These variables are listed in Table 3.3. For each variable, a brief description is included along with which of the TCD, WTC, or WA functions it is used in and the estimated size. The size estimates are given in terms of number of weapons (W), targets (N), and clusters (C). Scalar values are designated by “SC.” Based on the data structure and sizes involved and the type of computations required, the expected number of floating-point additions and multiplications, compares, operations per floating-point addition, operations per floating-point multiplication, and input/output operations were identified. Equations were derived to express the workload associated with each subfunction based on these operations and in terms of the mission variables N, W, and C. These equations were then encoded in the ADL files associated with the WTA/TS algorithm graphs and the library graph

Table 3.3. Data Variables Used in the WTA/TS Algorithm

	TCD	WTC	WA	Description
$[d_{im}]$	NxN	-	-	Dimension matrix used in TCD
N	sc	sc	sc	Number of targets ($i=1,\dots,N$)
μ_i	N	C	N_c	LaGrange multipliers
k	sc	sc	sc	Loop variable
α^k	50	50	50	Multiplier used in GLM
C	sc	sc	sc	Number of clusters ($c=1,\dots,C$)
$[X_{im}]$	NxN	-	-	Assignment matrix used in TCD
W	-	sc	sc	Number of weapons ($j=1,\dots,W$)
$[C_{xy}]$	NxN	CxW	$N_c \times W_c$	Temporary matrix used in GLM/ *Formation
$[m_i] = M$	N	C	W_c	Indexes used in *Formation
G_c	C	C	-	Set of targets assigned to cluster c
V_i	N	-	N_c	Target Value Vector
$[P_{ij}]$	-	NxW	$N_c \times W_c$	Kill probability matrix
$[T_{ij}]$	-	NxW	-	Target processing time matrix
$[S_{ij}]$	-	NxW	-	Weapon slew time matrix
$[A_{ij}]$	-	NxW	-	Average time available matrix
N_c	C	C	C	Number of target assigned to cluster c
R	-	sc	-	Redundancy of number weapons to a cluster
$[P_{cj}]$	-	CxW	-	Aggregated Kill probability matrix
$[T_{cj}]$	-	CxW	-	Aggregated Target processing time matrix
$[S_{cj}]$	-	CxW	-	Aggregated Weapon slew time matrix
$[A_{cj}]$	-	CxW	-	Aggregated Average time available matrix
V_c	-	C	-	Aggregated Target Value Vector
$[u_{cj}]$	-	CxW	-	Utilization matrix
u_{des}	-	sc	-	Average weapon utilization
$[Q_{cj}]$	-	CxW	-	Coefficient for Optimization
$[y_{cj}]$	-	CxW	-	Matrix for assigning weapons to clusters
$[Z_{cj}]$	-	CxW	-	Coefficient used in WTC
L_{wc}	-	C	C	Set of weapons assigned to cluster c
$[S_{ijc}]$	-	-	$N_c \times W_c$	Weapon slew time matrix
$[z_{ijc}]$	-	-	$N_c \times W_c$	Assigns weapon j to target i in cluster c
$[B_{ijc}]$	-	-	$N_c \times W_c$	Coefficient used in optimization
L_{wtc}	N_c	-	N_c	Targets assigned to each weapon in cluster c

Note: sc denotes a scalar value.

elements, along with equations that reflect characteristics of the hardware on which the operations would be executed.

Finally, those functions and operations that could be implemented in parallel were identified. The workload equations were modified to include a factor representing the degree of parallelism and, where necessary, alternate graphical descriptions were created to reflect different parallel approaches.

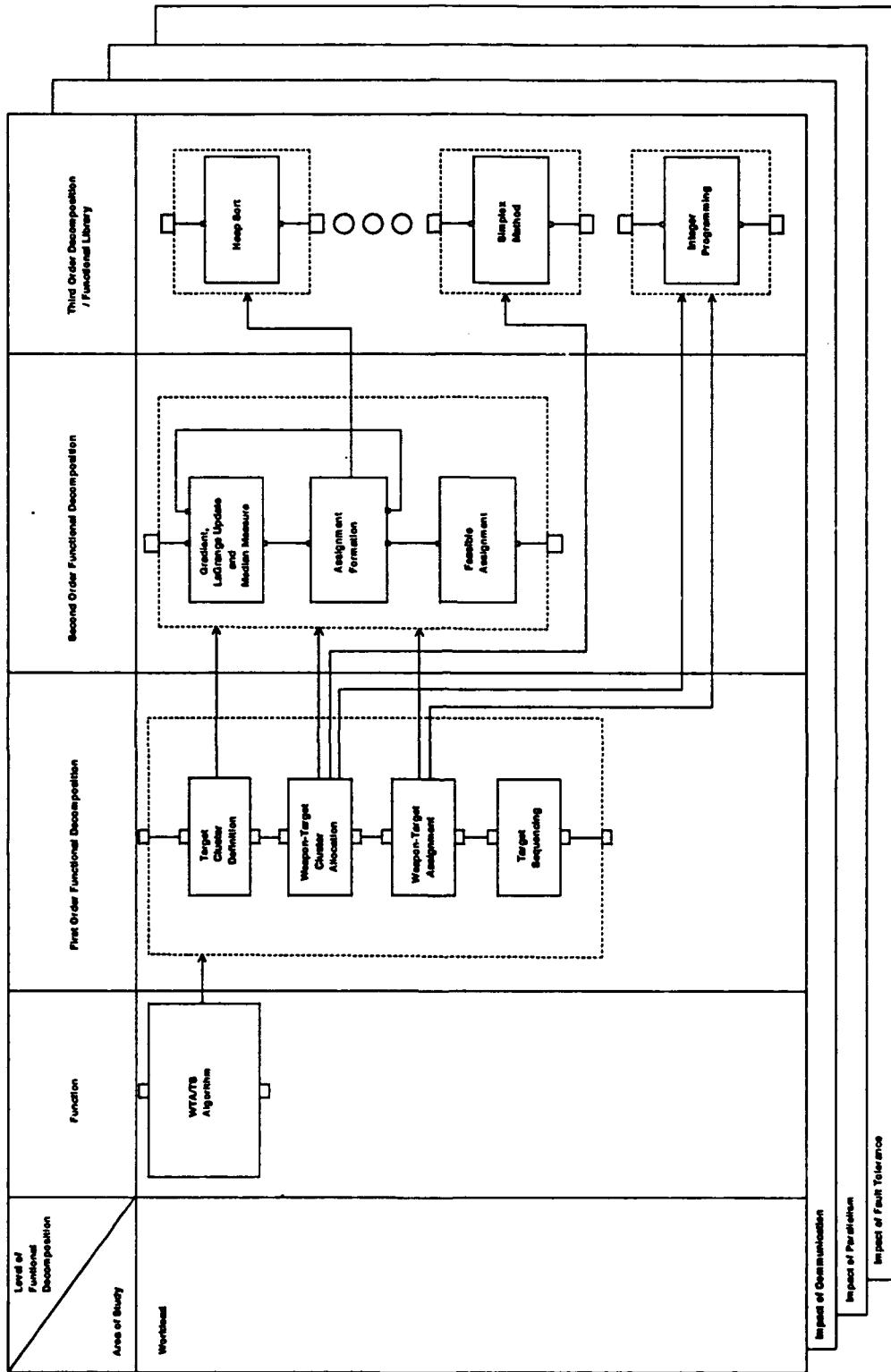
3.2.1.2.2. Library Structure

Figure 3.4 depicts, in an ADAS graph representation, the structure of the WTA/TS model incorporating the function library. This multidimensional diagram reflects the need for the model to be able to represent different aspects of system behavior and for the library to contain elements that support the different representations. This multiple representation enables the designer to consider the impacts of function decomposition, communication considerations, parallelism, and fault tolerance on a system's performance using the same model base instantiated with appropriate attribute values and subgraphs.

The hierarchical nature of the ADAS graph and subgraph representations and the ADL descriptions allow such a multiple representation. In Figure 3.4, a graph of the WTA/TS algorithm is decomposed into its four largest components. Three of these components, Target Cluster Definition (TCD), Weapon-to-Target Cluster Allocation (WTC), and Weapon-to-Target Assignment (WA), share in their respective subgraphs the three component functions shown in the next level of functional decomposition. These three components are the Gradient, LaGrange Update and Median Measure (GLM); the Assignment Formation; and the Feasible Assignment. The lowest level of functional decomposition consists of common functions which were selected for library modules. As indicated in the figure, these library modules are shared across multiple levels of decomposition. Shown at this level are the Heap Sort, the Simplex Method, and the Integer Programming modules.

Each ADAS graph component has a corresponding ADL file containing formulas to instantiate the model's attributes, based on mission scenario parameters and characteristics of the targeted hardware architecture.

The third dimension of Figure 3.4 represents the inclusion of alternate subgraphs to transform the model for system considerations such as architecture-specific parallelism and the resulting communication costs or fault tolerance schemes (e.g., voting). The alternate subgraphs and the ADL descriptions facilitate trade-offs in precision, speed, and architectures.



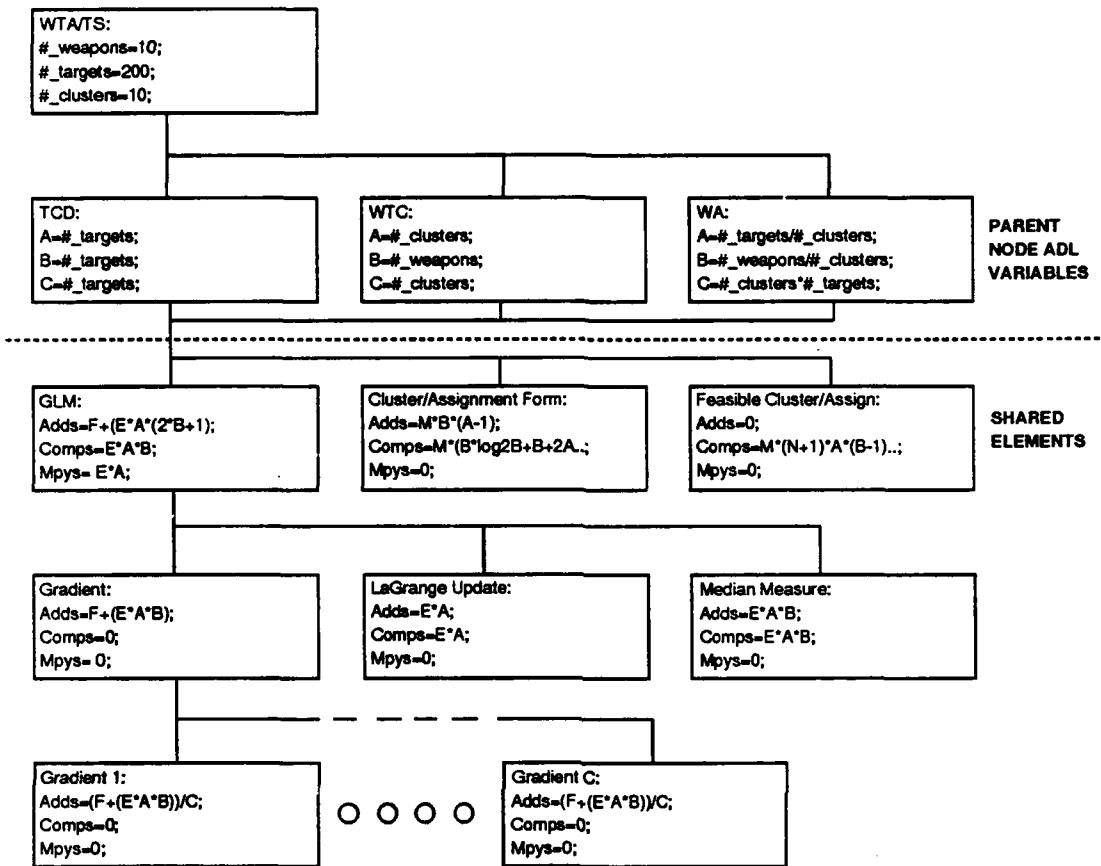


Figure 3.5. Shared Library Elements

3.2.1.2.3. Demonstration of Library

The use of the library was demonstrated in a case study to compare the performance of the Phase I and Phase II versions of the WTA/TS algorithm to see how the more detailed description of the algorithm influenced the performance predictions. The Phase I ADAS model did not develop the operation counts for constituent functions to the level developed for the common function library modules. In Phase I, all operation counts were directly dependent upon the number of targets, number of weapons, number of clusters, or some other top-level parameter. To incorporate the use of the common library modules, intermediate variables were created, as illustrated in Figure 3.5 to correlate variables in the module with the appropriate parameters at the higher level.

The attributes of the WTA/TS graph hierarchy were set through the invocation of

ADLEVAL. The attribute values were determined by equations in the ADL files associated with each graph element and library module. The equations reflected the expected number of floating-point additions and multiplications, compares, operations per floating-point addition, operations per floating-point multiplication, input/output operations, and degree of parallelization associated with that operation. These parameters were used in determining the operation counts. Once the operation counts were determined they were used in similar equations that reflected the hardware on which this operation was to be executed. The result was the expected execution time of the operation.

3.2.1.3. Results

The results of the modeling of the Phase I and Phase II algorithms can be seen in Figures 3.6 and 3.7, respectively. Although the TCD remained the dominant top-level component of the WTA/TS model, there is a significant change in the role of both the WTC and WA components. This change is attributable to the additional information gathered through the more detailed Phase II analysis of the algorithm.

Since TCD is the dominant top-level component for both algorithms, the roles of its subcomponents were also examined. These are shown in Figures 3.8 and 3.9, respectively. Examination of these two plots revealed that the Cluster Formation (CFORM) was the dominant subcomponent of TCD in Phase I, while the Gradient, LaGrange Update and Median Measure (GLM) was dominant in Phase II. This difference was the result of the use in CFORM of an $N \log N$ sort from the function library in Phase II, whereas an N^2 sort had been used in Phase I. The increase in execution time of the GLM component between Phase I and Phase II also resulted from the Phase II revisions.

These results demonstrate the need to study the performance impact of alternative design decisions and to re-evaluate predicted performance as more information becomes available and design decisions are made. The existence of a library of models of relevant computing functions and a modeling structure that allows the incorporation of library models into developing models facilitates the necessary trade-off studies and model revisions.

3.2.1.4. Conclusions

Considerable effort is required to develop performance models of algorithms. The use of function library modules can greatly reduce the amount of work involved and can facilitate algorithm and architecture trade-off studies. Parameterized attributes allow

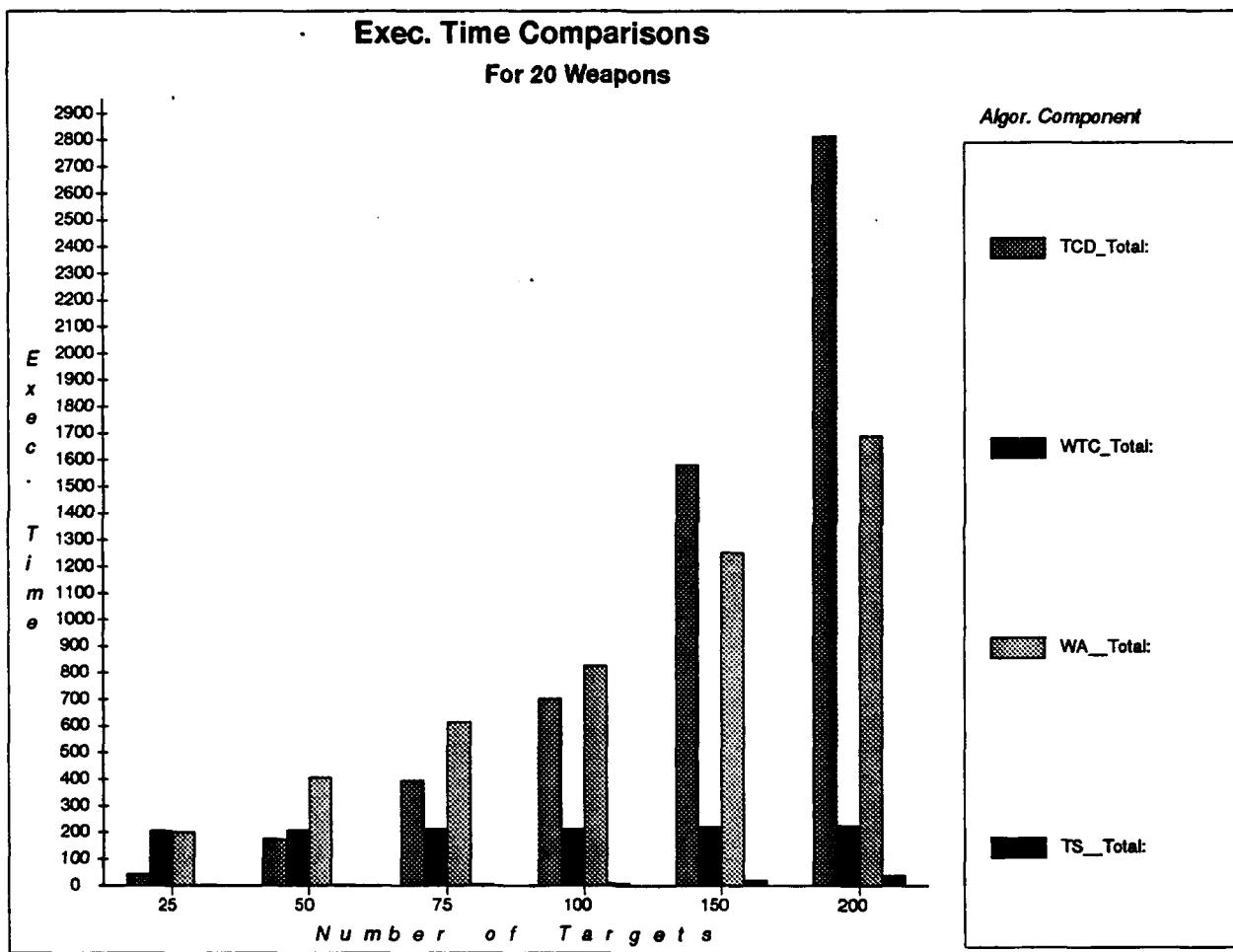


Figure 3.6. Phase I Top-Level Performance Analysis

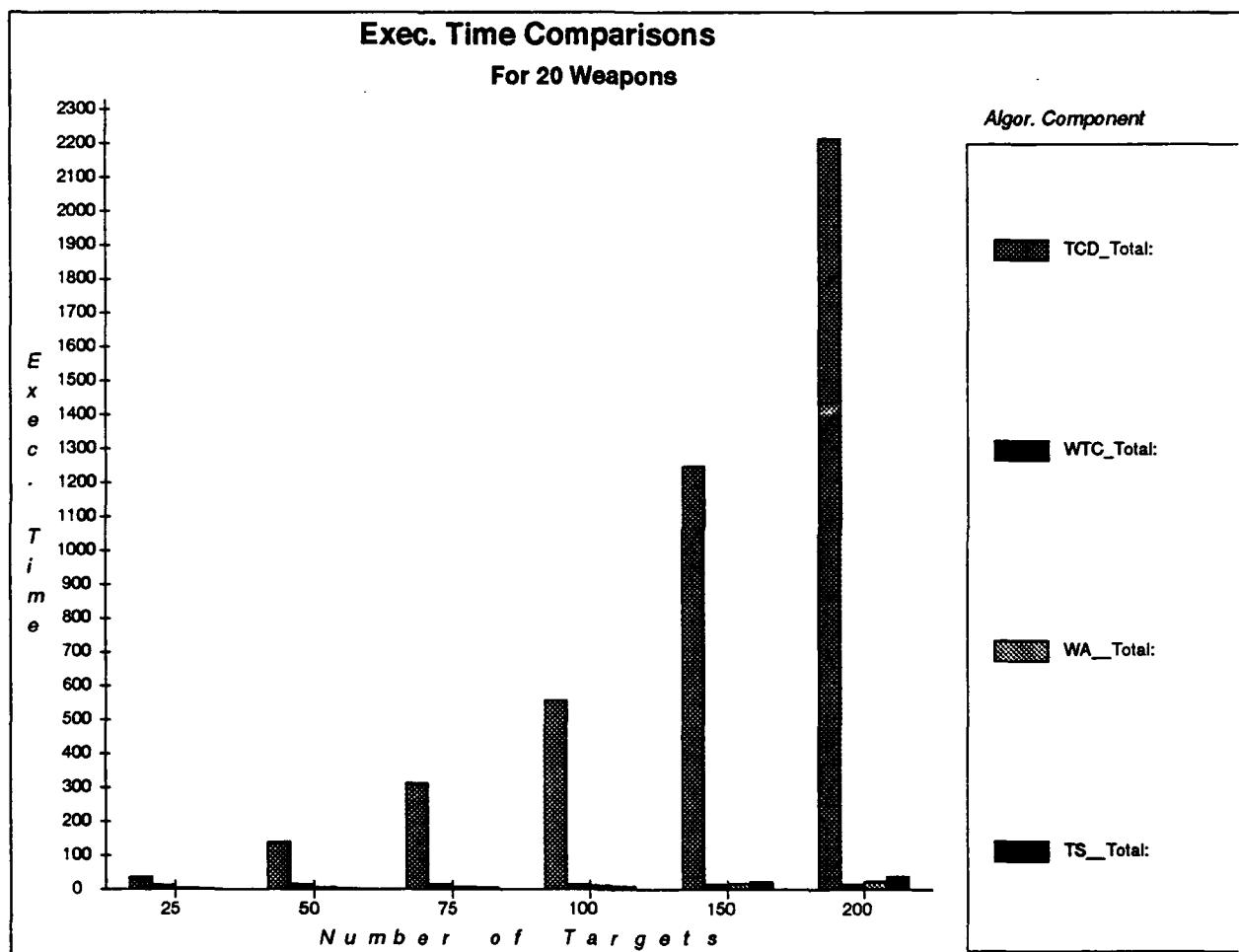


Figure 3.7. Phase II Top-Level Performance Analysis

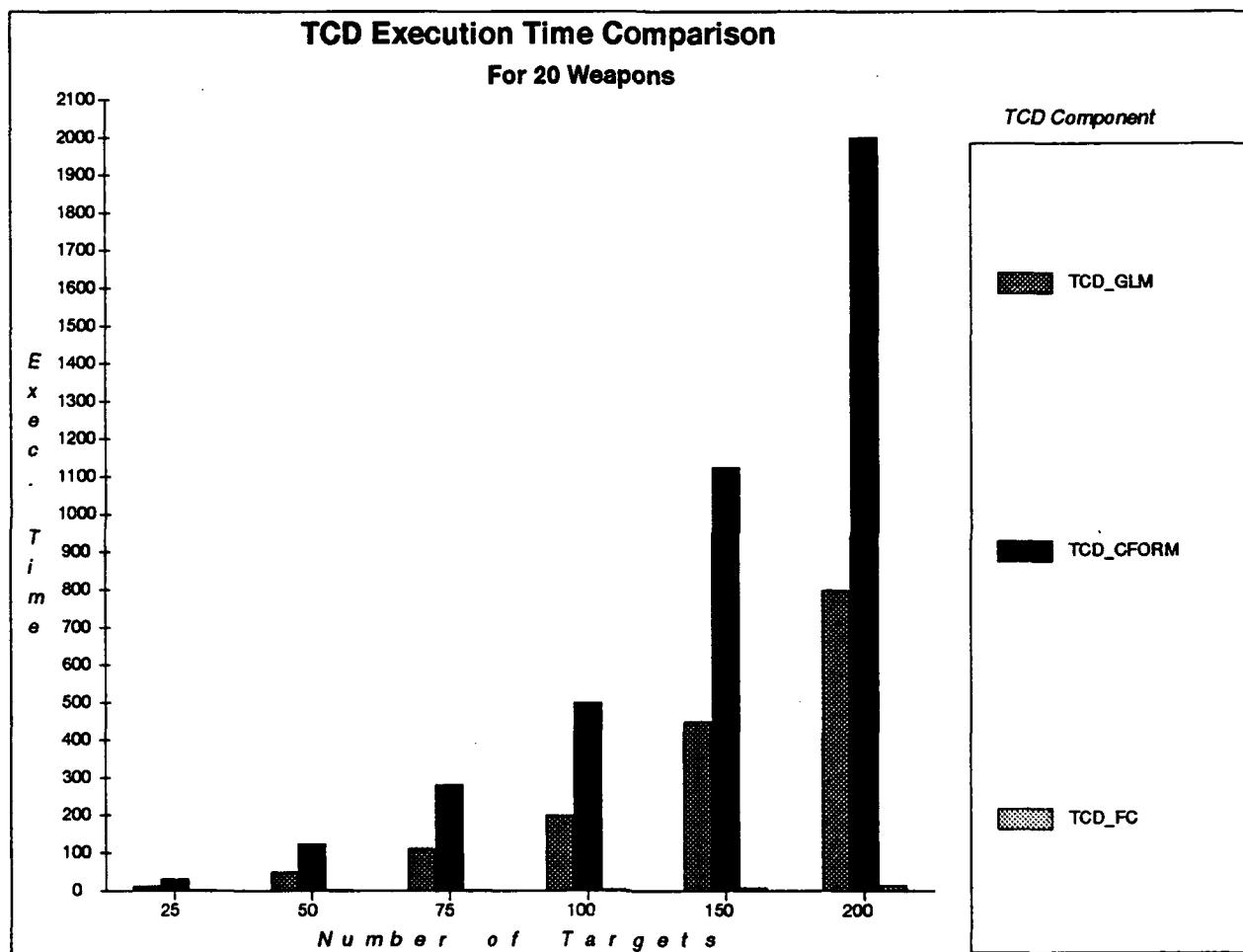


Figure 3.8. Phase I TCD Component Performance Analysis

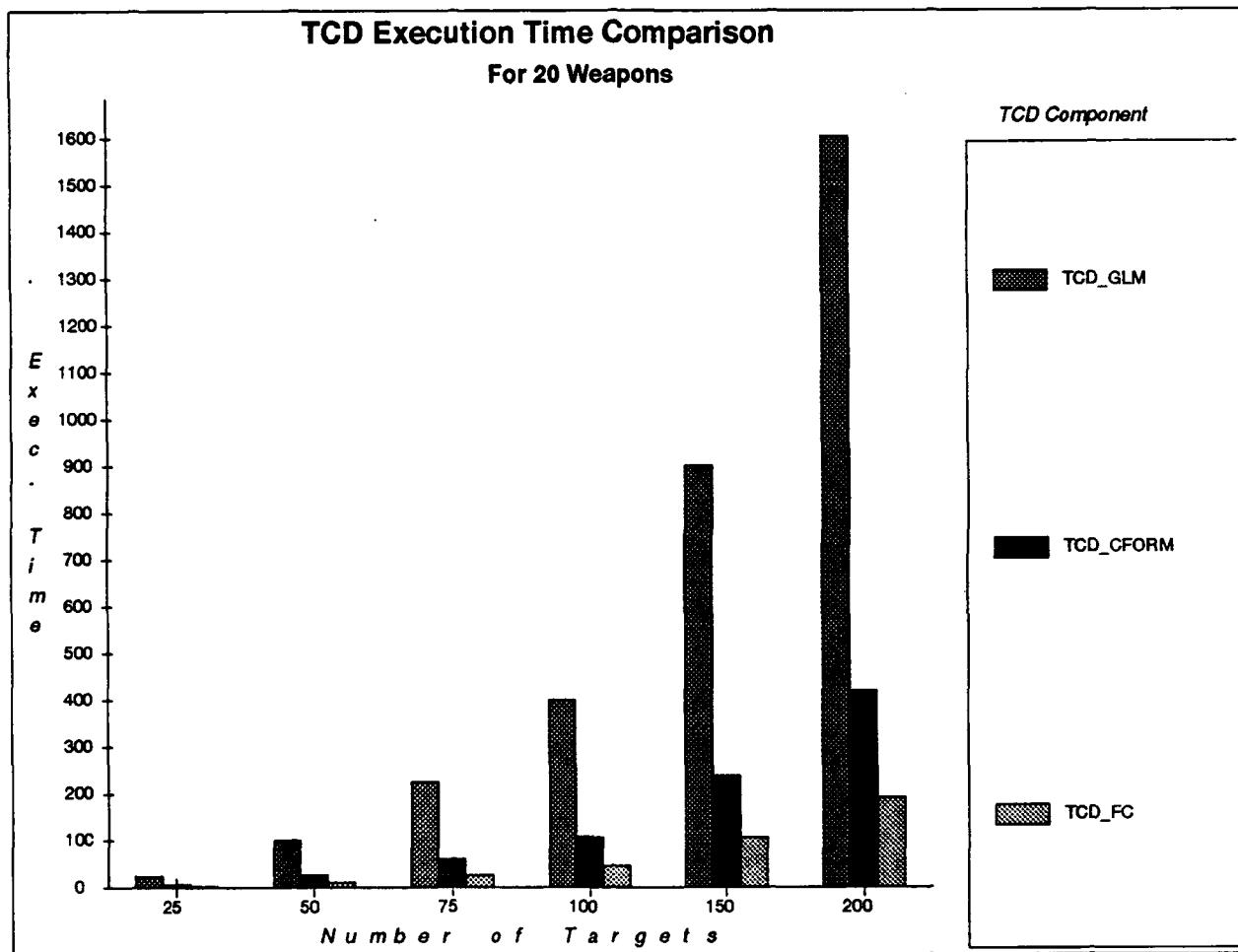


Figure 3.9. Phase II TCD Component Performance Analysis

models of functions to be built and allow different representations of model attributes. They also allow values in the model to be represented in terms of variable mission and system characteristics. These models are then generic in the sense that they can be used to model a broad range of applications by selecting different attribute representations and variable values. Thus, once a generic model has been built, it can be reused throughout the design cycle to evaluate different architectural designs or can be reused in the design of another application that uses the same algorithmic functions.

Generic software models can be particularly useful in the evaluation of parallel systems since multiple decompositions need to be considered as well as different architectures. A generic model of an algorithm can be instantiated and transformed to reflect these different implementations, and all models of that algorithm will originate from the same core model.

Generic models of frequently used functions can be built, validated, and stored in a function library. The library can also contain rules for decomposing and transforming the models with respect to system characteristics such as communication patterns, parallel resources, or fault tolerance.

3.2.2. High-Level Performance Assessment

3.2.2.1. Background

In Phase I of DAHPHRS, a model was created to examine the relationship between parallelism, processing and communication workloads, and processing speeds and communication speeds (bandwidths). Results indicated that a highly important factor in determining the speedup achievable through parallelism is the processing-workload-to-communication-workload ratio inherent in the algorithm or function. Therefore, if a systems designer knew this algorithmic ratio and its corresponding architectural characteristic, the processing-to-communication-speed ratio (PCSR), he could determine the degree of parallelism that would provide the maximum speedup. This case study was undertaken to determine if a static analysis of processing and communication balance would be useful in evaluating the effectiveness of different degrees of parallelism for an application.

3.2.2.2. Description

The approach taken was to identify the operations required to perform a row-by-matrix multiplication function; to create a processing/communication workload model of that function based on its operations; to define a static analysis technique that could evaluate the effectiveness of different degrees of parallelism for the workload model given different processing and communication speeds and to implement that analysis.

The matrix multiply function was described by the hierarchical ADAS graphs illustrated in Figures 3.10 through 3.12. Figure 3.10 contains the top-level graphs, while Figures 3.11 and 3.12 contain the nonparallel and parallel versions of the subgraph for the multiply node. Each node in these graphs represents the processing or communications workload necessitated by its constituent operations. The static analysis technique was based on computing performance delays for each node as the processing/communication workload to degree of parallelism to PCSR ratio and evaluating system performance attributes based on these ratios. This analysis was implemented by creating ADL files for the ADAS graphs and using ADLEVAL to compute the delays and evaluate performance based on equations and conditions defined in the ADL files.

The ADL description can also specify how the results of these calculations can be used to make recommendations. For example, Figure 3.13 contains an example of how the total estimated time a resource (ME, PE, or BUS) is busy can determine a recommendation of whether the degree of parallelism should be increased or decreased.

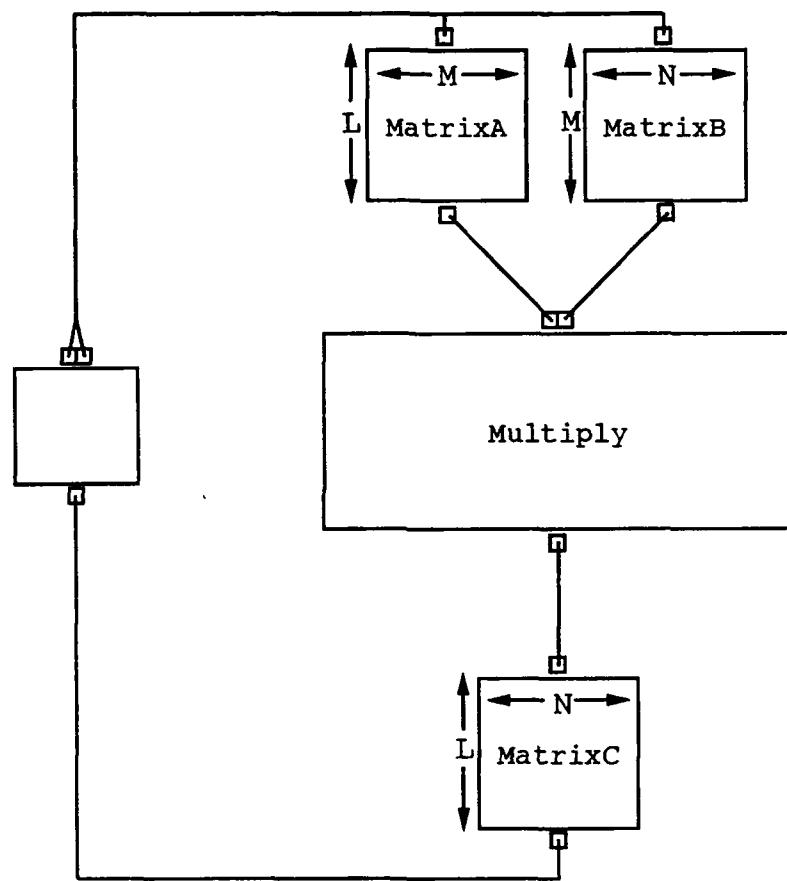


Figure 3.10. Top-Level ADAS Graph of Row-by-Matrix Matrix Multiply Function

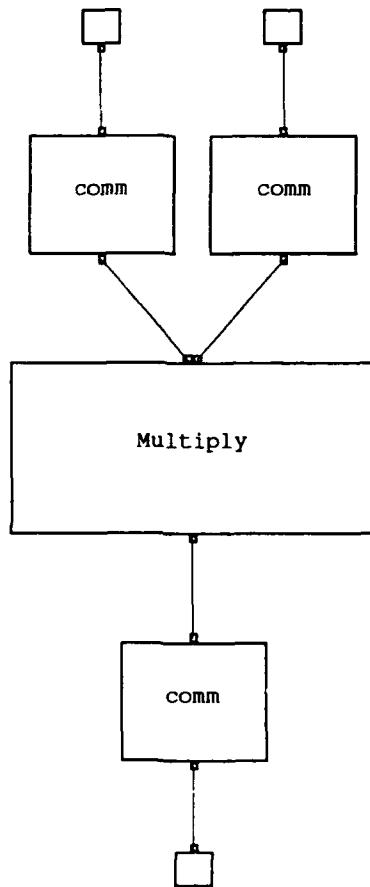


Figure 3.11. Non-Parallel ADAS Subgraph of Row-by-Matrix Matrix Multiply Function

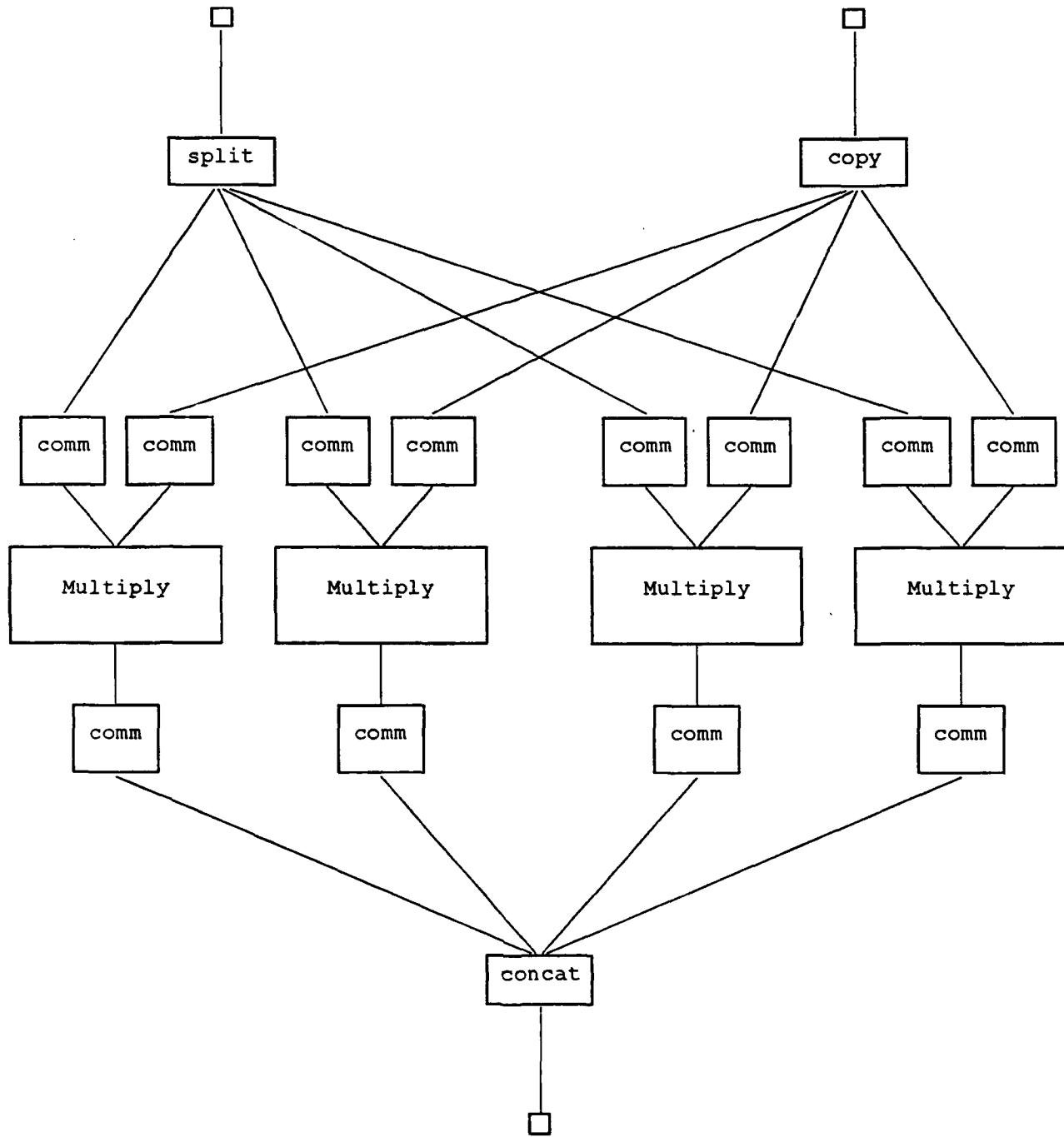


Figure 3.12. Parallel ADAS Subgraph of Row-by-Matrix Matrix Multiply Function

```

if (ME_Time_Busy >= BUS_Time_Busy) and
    (ME_Time_Busy >= PE_Time_Busy) then
    Recommendation =      "Decrease Parallelism";
elseif (ME_Time_Busy < BUS_Time_Busy) and
    (BUS_Time_Busy >= PE_Time_Busy) then
    Recommendation =      "Decrease Parallelism";
elseif (ME_Time_Busy < PE_Time_Busy) and
    (BUS_Time_Busy < PE_Time_Busy) then
    Recommendation =      "Increase Parallelism";
end if;

```

Figure 3.13. Parallelism Recommendation Code Written in ADL

3.2.2.3. Results

The results of the static analysis of the row-by-matrix function are listed in Table 3.4. This table contains recommendations to increase or decrease the degree of parallelism relative to the PCSR. A recommendation to increase parallelism is represented in this table by an “I” while a decrease parallelism recommendation is represented by a “D.” For example, if a row-by-matrix multiply were to be run on hardware whose PCSR was 10:1, the designer would want to use between six and eight processors to achieve maximum throughput for this function.

3.2.2.4. Conclusions

The use of static analysis of processing and communications balance within a system can be useful in determining an appropriate parallel decomposition of an algorithm. In combination with a function library of algorithm models and ADL descriptions, tables such as the one produced in this study could be made available for every element of the library. The designer, knowing the characteristics of the hardware to be used in the system, could then refer to the tables as a guide in choosing the degree of parallelism that would provide the highest throughput for the application function. The selected decomposition could then be examined in greater detail via simulation, where other factors that affect attainable speedup could be included.

Table 3.4. Parallelism Recommendation Matrix

Proc Speed	Comm Speed	PCSR	Parallelism Analyzed					
			1	2	4	6	8	16
1.00e01	1.00e03	1:100	I	I	I	I	I	I
1.00e02	1.00e03	1:10	I	I	I	I	I	I
6.25e02	1.00e03	1:1.6	I	I	I	I	I	I
1.00e03	1.00e03	1:1	I	I	I	I	I	I
1.25e03	1.00e03	1.25:1	I	I	I	I	I	I
2.50e03	1.00e03	2.5:1	I	I	I	I	I	I
5.00e03	1.00e03	5:1	I	I	I	I	I	D
1.00e04	1.00e03	10:1	I	I	I	I	D	D
2.00e04	1.00e03	20:1	I	I	I	D	D	D
4.00e04	1.00e03	40:1	I	I	D	D	D	D
8.00e04	1.00e03	80:1	I	D	D	D	D	D
1.00e05	1.00e03	100:1	I	D	D	D	D	D
1.60e05	1.00e03	160:1	I	D	D	D	D	D
1.00e06	1.00e03	1000:1	I	D	D	D	D	D
1.00e07	1.00e03	10000:1	I	D	D	D	D	D

3.2.3. Network Reliability Analysis

3.2.3.1. Introduction

The goal of this effort was to investigate the applicability of existing analytical techniques for network analysis to the analysis of communications networks for distributed architectures such as the Fault-Tolerant Parallel Processor (FTPP) and the Advanced Onboard Signal Processor (AOSP). The model for the topology of the network is assumed to be a *probabilistic graph* $G = (V, E)$, where V is a set of *nodes* (or *vertices*) and E is the set of undirected *edges*. The nodes represent the communication centers or concentrators in the network, while the edges represent the (in this case) bidirectional links. In addition to the nodes and edges, a probabilistic graph has a probability of operation associated with each node and each edge.

There are a number of basic assumptions inherent in this model. Besides omitting all information except that directly relating to the topology, the model assumes that the probabilities specified are statistically independent. Secondly, it is assumed that nodes do not fail, and hence all network failures are consequences of edge failures. (This assumption does not limit the class of problems under consideration; see section 1.4 of Colbourn [15]).

This work concerns the *all-terminal reliability* problem, the probability that every pair of nodes in the graph can communicate. This is the only problem for which reasonable bounds (in the general case) are available in sub-exponential time. This probability can be combined with the probability that the set of necessary nodes are operational in order to form a conservative estimate of the reliability of the system.

The complexity of most network reliability problems is such that they can be computed by counting Turing machines in polynomial time, denoted $\#P$ -complete, and thus are unlikely to have efficient solutions. For restricted classes of systems (for example, fully connected or series/parallel graphs), approximate solution algorithms can be determined. This implies that for all but small- to medium-size networks exact analysis of the reliability is not feasible, and approximations must be sought.

The first phase of this work consisted of a literature study, to determine the theoretical basis for and practical limitations of the existing analytical techniques. Simulation techniques were not considered in this study. The literature search included a study of Colbourn's *The Combinatorics of Network Reliability* and a review of recent articles [20-28]. Several classes of techniques were studied, and the reliability polynomial technique was chosen for further consideration.

Phase II then consisted of an application of the reliability polynomial method to the

8-node hypercube, the FTPP intercluster communication, and the AOSP bus interconnection network. The latter application required the extension of the reliability polynomial technique to incorporate the consideration of multistate failures.

3.2.3.2. Description

Most proposed analytic methods for dealing with the network reliability problem fall into one of four categories: exhaustive state enumeration, Boolean algebraic methods, the reliability polynomial, and Markov methods. Methods from each of the four categories are discussed briefly in the following sections. The reliability polynomial was selected as the method most applicable to the FTPP and the AOSP for this case study. Several techniques for bounding the reliability via approximate solution techniques are described.

3.2.3.2.1. Exhaustive State Enumeration

Exhaustive state enumeration is the easiest combinatorial solution method to understand and to implement. In this method, all possible states of the system are enumerated. Each state is classified as either an operational state or a failure state, and the probability for each state is determined. The probabilities for all the failure states are summed to form the unreliability of the system.

- Advantages

- It is simple to understand and implement.
- It is easily amenable to (bounded) truncation.
- It is applicable to multistate models (more than one failure mode).
- It can be combined easily with performance measures that can be expressed as a function of the current state, such as bandwidth or average delay.

- Disadvantages

- This is probably the least efficient of all methods used to analyze a system. However, the use of state truncation can enable analysis of relatively large systems (25–50 links) for short missions. Truncation of exhaustive state enumeration has been used [16] to quickly produce bounds on the reliability of the system for models of this size.

3.2.3.2.2. Boolean Algebraic Methods

These methods use minisets of the network (cutsets or pathsets) and then use Boolean algebra and set theory to predict the reliability of the network. An efficient implementation of the sum-of-disjoint-products (SDP) algorithm is described in [17].

- Advantages

- It is faster (significantly, although still exponential) than exhaustive state enumeration.
- Cutsets can be used (qualitatively) to analyze the system (to provide insights into the system operation).
- Performance measures can be included, if they are given as a function of a path (or a cut) rather than as a function of the system state.
- SDP methods can be truncated (with bounds).

- Disadvantages

- Generating all the cutsets or pathsets can be time consuming, but this process can be truncated as well, using the same method as was used in the exhaustive state enumeration truncation.

3.2.3.2.3. The Reliability Polynomial

A reliability polynomial in p (the reliability of an individual link) can be used when all the links in a network have the same failure probability.

- Advantages

- Very fast (linear in the number of links) bounds can be determined, once a few key parameters are determined. These key parameters can be determined in polynomial time.
- Performance measures can be incorporated easily if the performance metric is a function of the number of operational (or failed) links.
- The computations are symbolic in p . This means that the sensitivity of the result to the parameter p can be investigated very quickly.

- Disadvantages

- It can only be used if the links have the same (or nearly the same) probability of failure.

3.2.3.2.4. Markov Methods

- Advantages

- It is the most flexible and dynamic method.
- It can handle multiple failure modes.
- It can model fault/error recovery.
- It can incorporate performance measures.

- Disadvantages

- It is probably not feasible for systems with more than about fifty components, especially if the components have a significant probability of failure over the mission time.

3.2.3.2.5. Selected Method: The Reliability Polynomial

The technique selected for further study is the reliability polynomial [15], since it appears to be the one best suited for the intended applications. For the systems being considered, it is reasonable to assume that the links are identical with respect to failure probability, making the reliability polynomial a natural choice.

Suppose that the reliability of each of the m links is p . Then the reliability of the network is given by the polynomial

$$Rel(p) = \sum_{i=0}^m N_i p^i (1-p)^{m-i},$$

where N_i is the number of pathsets with i operational edges. Unfortunately, determining the complete set of the N_i coefficients is $\#P$ -complete. However, for the all-terminal reliability problem, one can determine bounds on the reliability polynomial in polynomial time by calculating only a few of the coefficients and estimating the remainder. Thus, an intractable problem can be reduced to a tractable one, and a close approximation to the desired result can be determined.

Suppose that the smallest pathset is of size ℓ , which means that at least ℓ of the links must be operational to connect the network (ℓ is usually at least $n - 1$, where n is the number of nodes in the network). This implies that there are no pathsets with fewer than ℓ links, so that all the $N_i, i \leq \ell - 1$, are zero.

Similarly, suppose that the smallest cutset is of size c , then all configurations with more than $(m - c)$ operational links are operational. This implies that the $N_i, i \geq m - c + 1$, are equal to the number of combinations of i links from m , or $\binom{m}{i}$.

In addition to c and ℓ , suppose that the number of pathsets of size ℓ (N_ℓ) and the number of cutsets of size c (N_{m-c}) are also known. (All of these parameters can be determined in polynomial time.) Although this information is not enough to determine the reliability polynomial exactly, there are several methods available for bounding the unknown coefficients. Each of the bounding methods is linear in m , and none is always better than the others.

The first method is the simplest bounding method, and results from assuming (conservatively) that the unknown coefficients are zero. A corresponding optimistic bound is realized by assuming that the unknown coefficients are all $\binom{m}{i}$. These *simple* bounds are thus given by [15]

$$\begin{aligned} Rel(p) &\geq N_\ell p^\ell (1-p)^{m-\ell} + N_{m-c} p^{m-c} (1-p)^c + \sum_{i=m-c+1}^m \binom{m}{i} p^i (1-p)^{m-i} \\ Rel(p) &\leq N_\ell p^\ell (1-p)^{m-\ell} + \sum_{i=\ell+1}^m \binom{m}{i} p^i (1-p)^{m-i} \end{aligned}$$

The second approach to bounding the unknown coefficients (the *BBST* bounds) assumes that the system is *coherent*. That is, it assumes that the failure of a link cannot render a failed state operational. (This is certainly not an unreasonable assumption.) This implies that every supergraph of an operational subgraph is itself operational, resulting in the following set of BBST bounds [15]:

$$Rel(p) \geq \sum_{i=0}^{c-1} \binom{m}{i} p^{m-i} (1-p)^i + N_{m-c} p^{m-c} (1-p)^c + \sum_{i=c+1}^{m-\ell} N_\ell \frac{\binom{m}{i}}{\binom{m}{m-\ell}} p^{m-i} (1-p)^i$$

$$Rel(p) \leq \sum_{i=0}^{c-1} \binom{m}{i} p^{m-i} (1-p)^i + N_\ell p^\ell (1-p)^{m-\ell} + \sum_{i=c}^{m-\ell-1} N_{m-c} \frac{\binom{m}{i}}{\binom{m}{c}} p^{m-i} (1-p)^i$$

The third set of bounds was derived by using set theoretic notions to improve on the BBST bounds defined above. The resulting bounds are called the *Kruskal-Katona* (K-K) bounds, and are given by [15]

$$\begin{aligned} Rel(p) &\geq \sum_{i=0}^{c-1} \binom{m}{i} p^{m-i} (1-p)^i + N_{m-c} p^{m-c} (1-p)^c + \sum_{i=c+1}^{m-\ell} N_\ell^{i/(m-\ell)} p^{m-i} (1-p)^i \\ Rel(p) &\leq \sum_{i=0}^{c-1} \binom{m}{i} p^{m-i} (1-p)^i + \sum_{i=c}^{m-\ell-1} N_{m-c}^{i/c} p^{m-i} (1-p)^i + N_\ell p^\ell (1-p)^{m-\ell} \end{aligned}$$

Each of these three bounds work best when p is either close to zero or close to one. The bounds are very fast to calculate, given the four input parameters (ℓ , c , N_ℓ and N_{m-c}). Given that no set of bounds can be shown to be always better than the others, in this study we calculated all three sets of bounds and then selected the maximum of the three lower bounds and the minimum of the three upper bounds as the lower and upper bounds for the reliability polynomial.

3.2.3.3. Results

The reliability polynomial bounding techniques were applied to the FTPP and the AOSP networks, since these networks are representative of the systems of interest. The models derived for these networks and the bounds that were computed are described in the following sections.

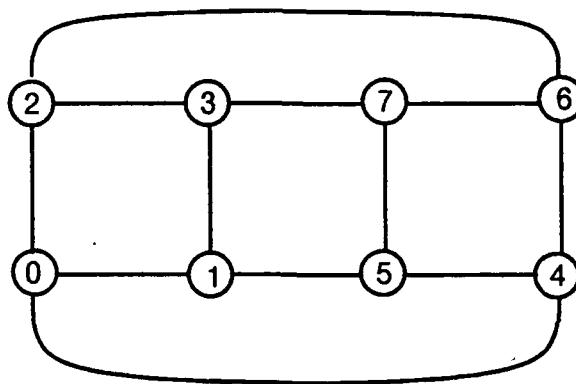


Figure 3.14. Eight-Node Hypercube Used for Example Solution

Table 3.5. Comparison of Low and High Estimates for Unknown Coefficient

Technique	N_8 Low Value	N_8 High Value
Exact Value	408	408
Simple Bounds	0	$\binom{12}{8} = 495$
BBST Bounds	$N_7 \binom{\frac{12}{4}}{\frac{12}{5}} = 240$	$N_9 \binom{\frac{12}{4}}{\frac{12}{3}} = 477$
K-K Bounds	$(N_7)^{\frac{4}{5}} = 116.8$	$(N_9)^{\frac{4}{3}} = 1264.1$

3.2.3.3.1. Eight-Node Hypercube

The 8-node hypercube, shown in Figure 3.14, has only 12 links, so it is easily amenable to exact solution. For this model all three bounds were calculated. The maximum lower bound and the minimum upper bound were selected and compared with the exact answer. The smallest cutset must be of size 3, since that is the number of failed links needed to disconnect any given node from the rest of the network. There must be at least seven operational links to connect all eight nodes. For the three bounds, we thus need the N_7 and the $N_{12-3} = N_9$ terms; the bounds techniques all estimate the N_8 terms from these. (All the N_i terms for $i \leq 6$ are zero, while the terms for $i \geq 10$ are $\binom{m}{i}$). Table 3.5 compares the lower bound and upper bound estimates for the three bounding techniques with the exact value for N_8 .

All three bounding techniques for the reliability estimate were compared with the exact answer. In this case, the BBST bounds gave the best lower and upper bounds. The BBST bounds can be seen in Table 3.5 to be better than the others since the estimates for N_8 are closer to the exact values. Table 3.6 shows the maximum lower

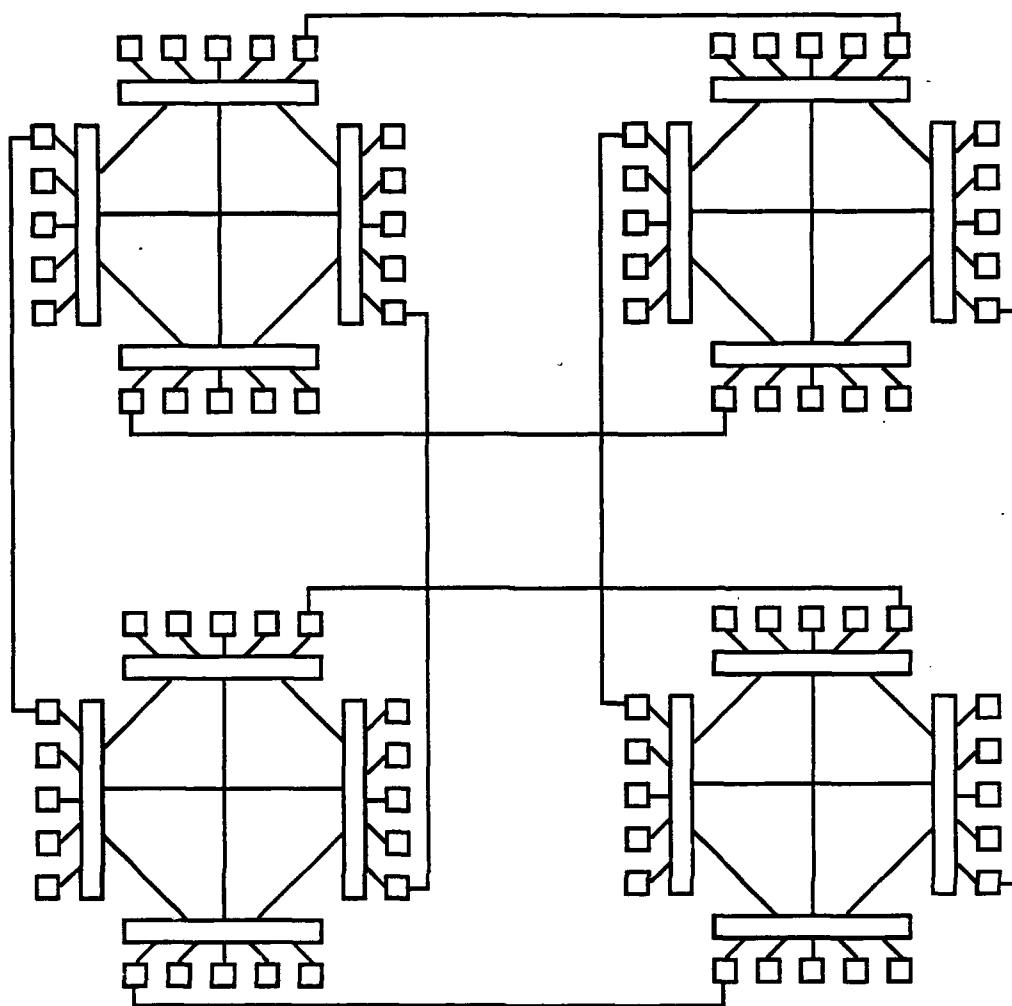


Figure 3.15. First Example FTPP System Analyzed

bound, the exact value, and the minimum upper bound for values of p between 0.8 and 1.0.

3.2.3.3.2. FTPP Interconnection Network

The applicability of the reliability polynomial technique to the intercluster communications network for several examples of the FTPP system was investigated next. The first instance of the FTPP investigated is shown in Figure 3.15. There are a total of 16 nodes and 32 links in the communication subsystem if each network element (NE) and its associated I/O processor are considered to be a node. The network elements within a cluster are fully connected, and these intracluster links are often included in an analysis of the reliability of a cluster. For this reason, a cluster (four

Table 3.6. Comparison of Bounds with Exact Reliability for 8-Node Hypercube

p (Link Reliability)	Lower Bound	Exact Value	Upper Bound
0.800	0.87617332	0.89550072	0.93979251
0.805	0.88417917	0.90329778	0.94460928
0.810	0.89192539	0.91074347	0.94915789
0.815	0.89940655	0.91783732	0.95344412
0.820	0.90661752	0.92457998	0.95747435
0.825	0.91355407	0.93097323	0.96125561
0.830	0.92021185	0.93701971	0.96479481
0.835	0.92658752	0.94272298	0.96809971
0.840	0.93267804	0.94808739	0.97117817
0.845	0.93848127	0.95311862	0.97403860
0.850	0.94399559	0.95782292	0.97668934
0.855	0.94922006	0.96220738	0.97913927
0.860	0.95415443	0.96628004	0.98139703
0.865	0.95879924	0.97004968	0.98347181
0.870	0.96315598	0.97352564	0.98537248
0.875	0.96722698	0.97671843	0.98710865
0.880	0.97101492	0.97963846	0.98868895
0.885	0.97452396	0.98229718	0.99012268
0.890	0.97775918	0.98470646	0.99141878
0.895	0.98072588	0.98687863	0.99258608
0.900	0.98343098	0.98882622	0.99363309
0.905	0.98588210	0.99056208	0.99456835
0.910	0.98808777	0.99209929	0.99540001
0.915	0.99005741	0.99345124	0.99613589
0.920	0.99180132	0.99463093	0.99678338
0.925	0.99333084	0.99565190	0.99734992
0.930	0.99465823	0.99652708	0.99784249
0.935	0.99579602	0.99726939	0.99826711
0.940	0.99675810	0.99789178	0.99863040
0.945	0.99755859	0.99840629	0.99893785
0.950	0.99821228	0.99882507	0.99919498
0.955	0.99873418	0.99915957	0.99940658
0.960	0.99914002	0.99942076	0.99957770
0.965	0.99944526	0.99961907	0.99971271
0.970	0.99966526	0.99976444	0.99981570
0.975	0.99981564	0.99986601	0.99989122
0.980	0.99991071	0.99993253	0.99994296
0.985	0.99996471	0.99997199	0.99997532
0.990	0.99999028	0.99999183	0.99999249
0.995	0.99999887	0.99999899	0.99999905
1.000	1.00000000	1.00000000	1.00000000

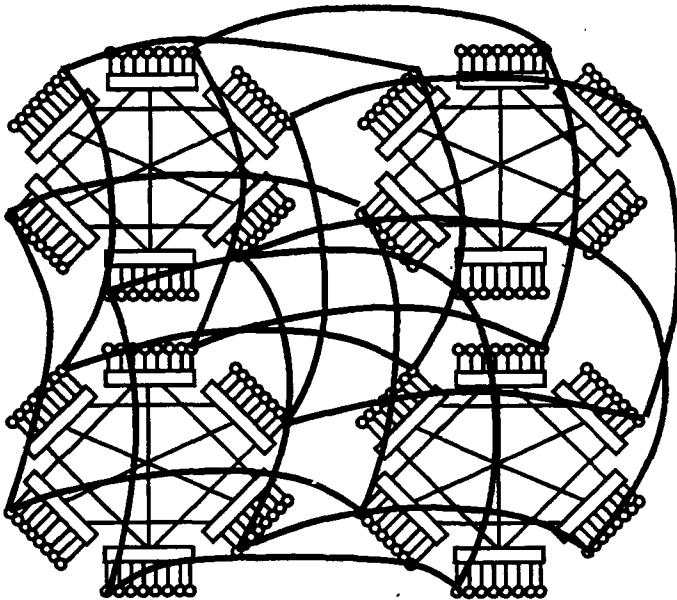


Figure 3.16. Second Example FTPP System Investigated

network elements, the attached processing elements, and the intracluster links) will be considered as a node in the graph, rather than each NE, and the intercluster links are the links in the graph. This results in a system with only 4 nodes and 8 links, a system that is small enough to be solved exactly. The reliability of the intercluster communication subsystem for the first example is then given by

$$Rel(p) = 32p^3(1-p)^5 + 64p^4(1-p)^4 + 56p^5(1-p)^3 + 28p^6(1-p)^2 + 8p^7(1-p) + p^8$$

A larger system, described in [18] and shown in Figure 3.16, was considered next. This system again has 4 nodes, but has 24 intercluster links. There are six ringlike structures of links connecting the clusters; each NE in a cluster is connected to the similarly positioned NE in the neighbor cluster in the clockwise direction and to the similarly positioned NE in the neighbor cluster in the counterclockwise direction. Since there are then 12 links connected to each cluster, the smallest cutset is of size 12. Three links are necessary to connect the 4 nodes, so the smallest pathset is size 3. For this configuration, the number of pathsets of size 3 is 864 (out of a maximum possible of 2024) and the number of cutsets of size 12 is 2,704,150 (out of a maximum possible of 2,704,156). The greatest lower and least upper bounds on the reliability (from the three techniques) are given in Table 3.7.

The next study investigated a minor architectural alternative for the interconnection topology using the same set of nodes and the same number of links. Instead of using six isomorphic ringlike structures, the nodes were connected using two ringlike structures, two butterfly-like connections and two hourglass connections. That is,

Table 3.7. Reliability Bounds for Original Configuration of 4-Node, 24-Link FTPP

Individual Link Reliability, p	Maximum Lower Bound	Minimum Upper Bound
0.800	0.99987574	1.00000000
0.810	0.99992788	1.00000000
0.820	0.99995967	1.00000000
0.830	0.99997836	1.00000000
0.840	0.99998892	1.00000000
0.850	0.99999461	1.00000000
0.860	0.99999753	1.00000000
0.870	0.99999894	1.00000000
0.880	0.99999958	1.00000000
0.890	0.99999985	1.00000000
0.900	0.99999995	1.00000000
0.910	0.99999999	1.00000000
0.920	≥ 0.99999999	1.00000000
0.930	≥ 0.99999999	1.00000000
0.940	≥ 0.99999999	1.00000000
0.950	≥ 0.99999999	1.00000000
0.960	≥ 0.99999999	1.00000000
0.970	≥ 0.99999999	1.00000000
0.980	≥ 0.99999999	1.00000000
0.990	≥ 0.99999999	1.00000000

Table 3.8. Comparison of Coefficients for Original and Alternative Configurations

Coefficient	Original System	Alternative
N_3	864	1024
N_{12}	2,704,150	2,704,152

suppose that the four nodes are labeled N, S, E and W, giving relative locations on a compass. The ringlike structures included the following links: $N \leftrightarrow E$, $E \leftrightarrow S$, $S \leftrightarrow W$ and $W \leftrightarrow N$. The butterfly-like structures used the following links: $N \leftrightarrow S$, $W \leftrightarrow E$, $E \leftrightarrow S$ and $W \leftrightarrow N$. The hourglass structure used links between $N \leftrightarrow S$, $W \leftrightarrow E$, $N \leftrightarrow E$ and $S \leftrightarrow W$. This change did not alter the size of the minimum cutset nor the size of the minimum pathset, but it did increase the number of 3-link pathsets and decrease the number of 12-link cutsets, resulting in an increase in the reliability of the system. The N_t and N_{m-c} coefficients for the two alternatives are shown in Table 3.8. The effect on the reliability is minimal (i.e., too small to be reflected in a table); however, since the original configuration had such a high degree of redundancy, it had a very high reliability already. In the more general case, this example shows how the calculation of only two coefficients of the polynomial can be used to determine which system design alternative is more reliable.

The next study looked at the issue of redundant communications paths for the FTPP clusters. That is, suppose that three distinct (independent, nonoverlapping) paths are needed between each set of nodes rather than only one path. Assume that these three paths can occur on one of three "circuits," which are defined a priori. That is, the six ringlike structures are divided into three groups of two each, and the three distinct paths are generated one from each group. This consideration of triplex messages simplifies the analysis in that the reliability of each group of links can be analyzed separately, since they are independent.

Looking at the original configuration then, a group of links consists of 4 nodes and 8 links, where there are two connections each between the following pairs of nodes: $N \leftrightarrow E$, $E \leftrightarrow S$, $S \leftrightarrow W$ and $W \leftrightarrow N$. The reliability of each group or circuit is given by

$$Rel_g = 32p^3(1-p)^5 + 64p^4(1-p)^4 + 56p^5(1-p)^3 + 28p^6(1-p)^2 + 8p^7(1-p) + p^8$$

If three paths are needed, the combined reliability is just $(Rel_g(p))^3$, while if any two of the three paths are needed, the combined reliability is $3(Rel_g(p))^2 - 2(Rel_g(p))^3$.

One final alternative was considered. As in the most recent example, assume that there are three independent groups of links of which paths in either two or three are needed. Instead of providing 8 links in each group, consider using only 6 (for a total

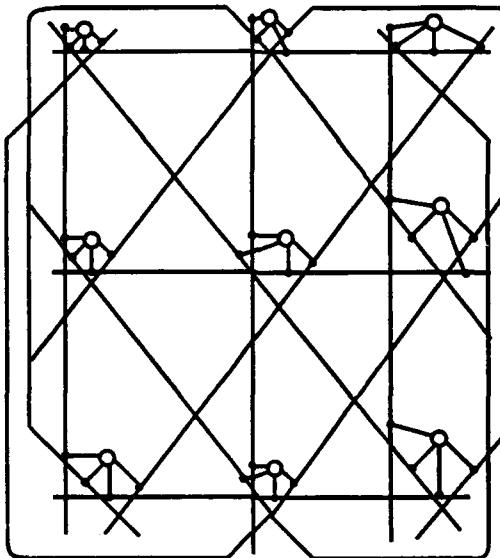


Figure 3.17. AOSP System

of 18 rather than 24). In addition to a single ring structure ($N \leftrightarrow E$, $E \leftrightarrow S$, $S \leftrightarrow W$, $W \leftrightarrow N$) add two cross links ($N \leftrightarrow S$ and $E \leftrightarrow W$). The reliability of this alternative is given by

$$Rel_g' = 16p^3(1-p)^3 + 15p^4(1-p)^2 + 6p^5(1-p) + p^6$$

Now, in comparing the two, $Rel_g > Rel_g'$, except for values of p greater than about 0.75 it is less than a 5% improvement. When $p \geq 0.9$ one realizes a less than 1% improvement with a 33% increase in hardware. Thus, it seems reasonable in this scenario to use only 18 links rather than 24.

In summary, several different interpretations of the FTPP intercluster communications network were modeled, and the reliability polynomial technique was applied to all cases. Several architectural alternatives could be compared using this method; in some cases a comparison was possible via the calculation of only a small number of coefficients of the reliability polynomial.

3.2.3.3.3. AOSP Bus Interconnection Network

Next, the reliability polynomial technique was applied to the interelement bus interconnection network for the AOSP, shown in Figure 3.17. Each node represents an Array Computing Element (ACE) [19] connected to four different buses. The requirements for the AOSP interconnection state that a node must be able to communicate through three of its four ports. The links to the buses (ports) can fail in one of two modes; the buses do not fail of their own accord. The first mode of link failure is

benign (it does not affect any other components), and has a probability of occurrence of $\alpha = 0.74$. The second mode has a complementary probability of occurrence ($\beta = 1 - \alpha = 0.26$), and is a malignant failure. A malignant failure on a port short circuits the bus and renders all the other ports connected to the same bus unusable. Since the system can be modeled by considering the links (ports) only, the model then has 9 nodes and 36 links. The consideration of multiple failure modes and dependent failures requires two modifications to the reliability polynomial method, since the reliability polynomial can model neither multiple failure modes nor dependent failures. When determining the N_ℓ and N_{m-c} terms, the first modification consists of examining each possible configuration state with ℓ or $m - c$ operational links for dependencies between the failed and non-failed components. The second modification alters the counting process for determining the N_ℓ and N_{m-c} terms by assigning weights to the count based on the relative probabilities of the different failure modes. These modifications result in an exact determination of the N_ℓ and N_{m-c} terms, and would result in an exact determination of all the other coefficients if the bounding techniques were not being used.

In the case of dependent failures, the failure of a link renders the other links on the same bus inoperative. To determine a coefficient (say N_i) of the reliability polynomial (whether or not there are dependent failures), enumerate all the possible configurations with i operational components and count the number that are operational. This count is then N_i . This counting procedure is altered in the following sections to account for dependent failures and multistate failures.

3.2.3.3.3.1. Dependent Failures

In the case where there are dependent failures, proceed as follows: suppose there is some configuration with i operational components and $m - i$ failed components. Before determining if the configuration represents an operational mode of the system, look at the $m - i$ failed components to see if any of the operational components are dependent on one of the failed components. If so, flag the dependent components as having failed and also determine if the configuration represents an operational mode of the system. If the system is operational, increment the counter for N_i . The N_i counter is incremented although there may actually be more than $m - i$ failed components (i.e., fewer than i operational components) after considering all the dependencies.

For example, suppose that there are three components, and the failure of the first causes the failure of the second. For the one-failure configurations, in the one in which the first component is down, flag the second component as having failed as well, before determining if the system is still operational. Since the failure of the first component caused the failure of the second (i.e., the second component did not fail

on its own), the probability associated with this configuration is $(1 - p_1)p_2p_3$ and not $(1 - p_1)(1 - p_2)p_3$. However, when looking at the two-failure configurations, the same configuration (both the first and second components failed) can be reached, but since the components have presumably failed independently, the probability for this same configuration is now $(1 - p_1)(1 - p_2)p_3$.

3.2.3.3.3.2. Multistate Failures

In the case of mutually exclusive multiple failure modes, given an operational configuration with i operational and $m - i$ failed links and two possible failure modes with probabilities α and β , the following procedure can be used to determine N_i : for each of the $m - i$ failed components, there are two different possibilities to consider, giving 2^{m-i} possible subconfigurations for each configuration. For each failed component, the first subconfiguration has probability α and the other has probability $\beta (= 1 - \alpha)$. Next consider each of the two subconfigurations and assign weights to them according to the probability of occurrence. The weight for the first failure is α while the weight for the second failure is β . If there is more than one failed component, then the weight is calculated by forming the product of the weights associated with each of the failed components. As an example, suppose that there are two failed components in a particular configuration being considered, and that each failure could be one of two types. Then there are really four different possibilities associated with this configuration, and their weights are α^2 , $\alpha\beta$, $\beta\alpha$ and β^2 . Some of these four subconfigurations might be operational, while some might represent system failure. For those that are operational, the weight value is added to the counter for N_i , rather than incrementing the counter by one. If all four sub-configurations are operational, then the sum of the four weights is added ($\alpha^2 + 2\alpha\beta + \beta^2 = 1$).

This technique results in non-integer values for N_i , but the bounding techniques are still applicable. A disadvantage of this approach is that the counts that result are particular to the chosen α and β ; however, the polynomial is still symbolic in p , the probability of link failure.

3.2.3.3.3.3. Application to AOSP

Since the malignant failure mode for the AOSP causes dependent failures, the two techniques described above are combined using the following procedure: for each configuration with $m - i$ failed components, consider all the combinations of failure modes (subconfigurations) and calculate a cumulative weight for each subconfiguration by multiplying the weights associated with the failure mode of each failed component. If the subconfiguration contains any malignant failures, check for dependent failures.

Table 3.9. Reliability Bounds for AOSP Network

p	Lower Bound	Exact	Upper Bound
0.9500	0.64360670	0.7641634	0.82404867
0.9525	0.66917989	0.7435724	0.83314078
0.9550	0.69478443	0.7629066	0.84257148
0.9575	0.72031163	0.7821086	0.85232254
0.9600	0.74564314	0.8011155	0.86236958
0.9625	0.77065100	0.8198587	0.87268116
0.9650	0.79519783	0.8382636	0.88321774
0.9675	0.81913723	0.8562493	0.89393053
0.9700	0.84231440	0.8737277	0.90476020
0.9725	0.86456694	0.8906032	0.91563540
0.9750	0.88572611	0.9067727	0.92647121
0.9775	0.90561825	0.9221240	0.93716733
0.9800	0.92406678	0.9365364	0.94760604
0.9825	0.94089456	0.9498792	0.95765010
0.9850	0.95592685	0.9620115	0.96714018
0.9875	0.96899492	0.9727811	0.97589222
0.9900	0.97994030	0.9820244	0.98369441
0.9925	0.98861994	0.9895651	0.99030384
0.9950	0.99491226	0.9952132	0.99544282
0.9975	0.99872429	0.9987646	0.99879483
1.0000	1.00000000	1.00000000	1.00000000

If the resulting configuration is operational, add the weight value to the counter for N_i .

For the AOSP network, there are a total of 9 nodes and 36 links, and each node must have three of the four links operational in order for the system to be considered operational. The failure of any two links on the same node can kill the system; the smallest cutset is of size 2. Since all 9 nodes must have three operational ports each, the smallest pathset is of size 27. The coefficients associated with these points, given $\alpha = 0.74$ and $\beta = 0.26$ and assuming that a malignant failure disables all the ports connected to the same bus, are then $N_{27} = 17443.17$ and $N_{34} = 425.96$. The unreliability bounds are listed in Table 3.9. Also listed in Table 3.9 is the exact result determined from the calculation of all the coefficients of the polynomial using the method described above.

3.2.3.4. Conclusions

The reliability polynomial technique has been applied to several different systems that were considered in this report. The 8-node hypercube was modeled exactly using the polynomial, and by using three bounding techniques. Additionally, several different scenarios for the intercluster communications for the FTPP were modeled. The reliability polynomial proved adequate for this task as well. The AOSP presented two modeling difficulties since the failures of all components were not independent and there were two possible different failure modes. The reliability polynomial technique was adapted to include consideration of both of these system aspects and reasonable bounds on the reliability of the system were derived.

4. Initial Design Modeling Phase

4.1. Description

When the baseline characteristics of the system have been established and initial assumptions have been made regarding the fault detection, isolation, and recovery (FDIR) mechanisms, the initial design modeling phase can be initiated. This phase corresponds with iterations through the Working Group design steps as illustrated in Figure 4.1, where alternative system designs are developed from the high-level designs and trade-offs between them are conducted to select a candidate architecture. In this phase, fault tolerance evaluation can be more quantitative and the quantitative analyses can be more thorough. The focus shifts to establishing the viability of particular designs and making finer discriminations between them.

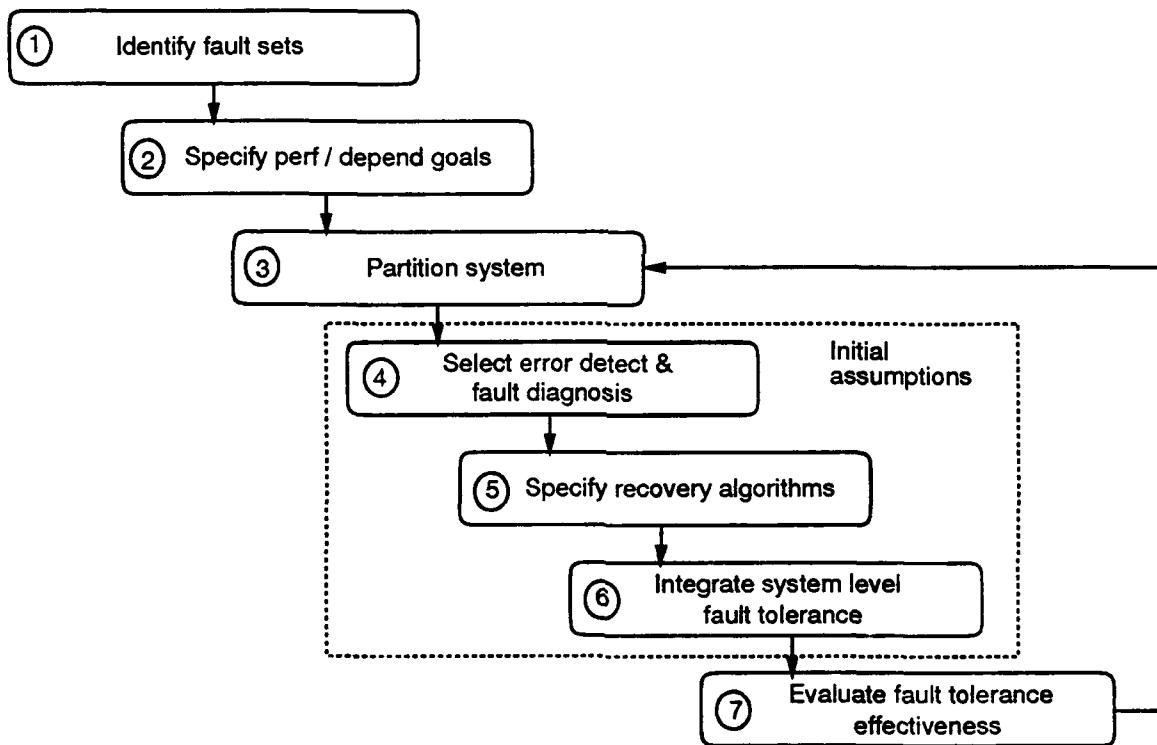


Figure 4.1. Initial Design Relationship to Methodology

The initial design modeling process is illustrated in Figure 4.2. As indicated in this figure, the algorithm descriptions from the baseline determination modeling phase are required as well as the mission requirements. It is also required that initial

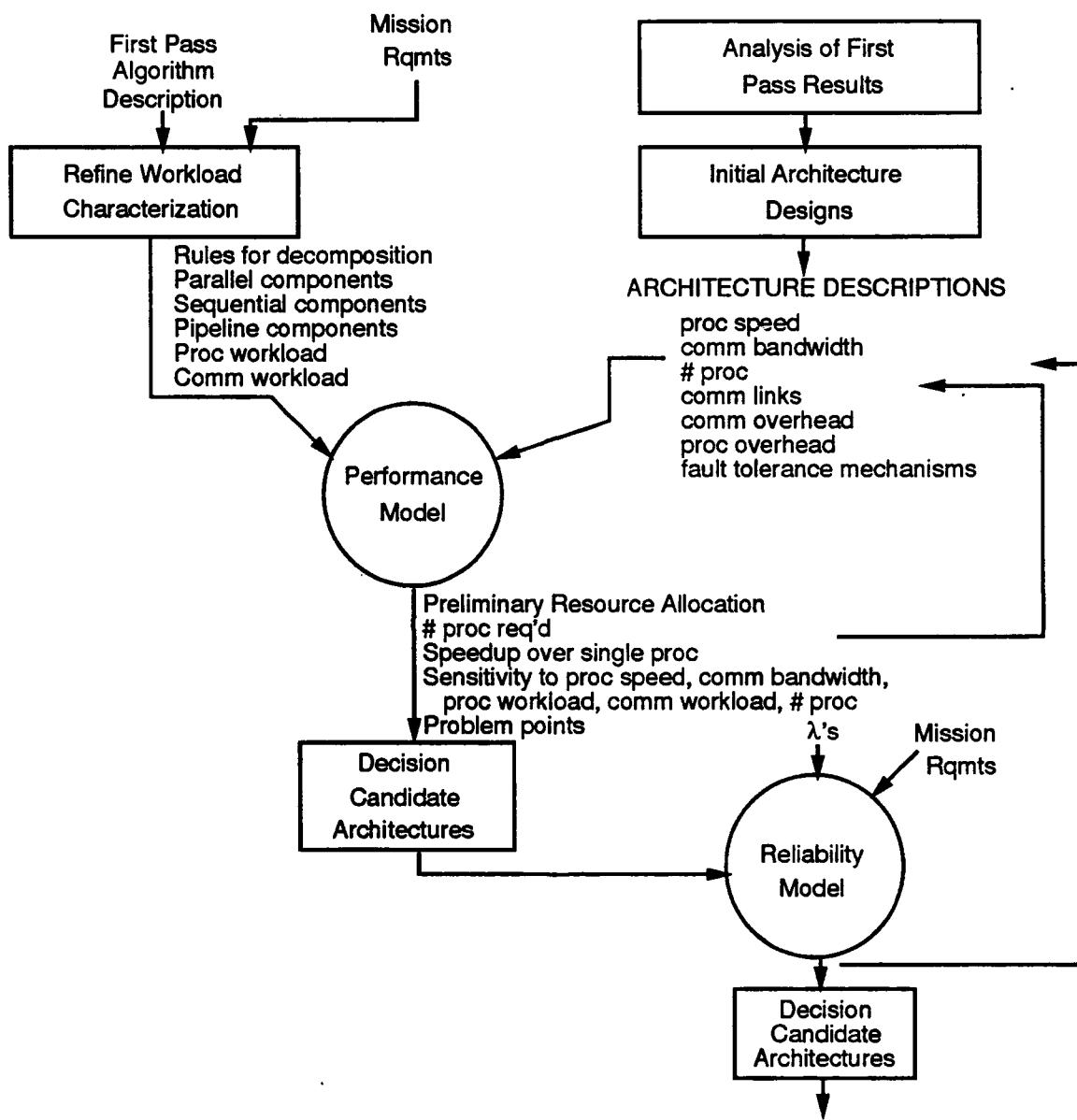


Figure 4.2. Initial Design Phase Process Diagram

architecture designs be consolidated from the results of the baseline determination phase. This modeling phase requires that the previous workload characterization be refined and developed to a lower level of detail to reflect the increasing design detail available for the algorithms. It has to address rules for decomposing the algorithm into parallel and sequential components and provide a breakdown of workload into processing and communication components.

From the initial architecture designs, the architecture models from the baseline determination phase can be refined to include more detail about the architecture, including more accurate estimates of architectural parameters such as processing speed, communications bandwidth, and processing and communications overhead. The models also have to be able to capture the functionality of architectural and algorithm components. This requires that the high-level performance models constructed during the baseline determination phase be expanded to include the functional models of Figure 2.8. It is also advantageous to have the engineering models of Figure 2.8 for the application functions to provide more accurate data for the workload characterization. Since the initial assumptions regarding the FDIR mechanisms have been made, the architecture models can also be refined to include components that implement those mechanisms.

For each of the candidate architectures, reliability models can be constructed in accordance with the reliability analysis requirements previously established and those requirements can be refined in accordance with the proposed FDIR mechanisms and architecture characteristics. At this stage it is necessary that a parameterized model be created that incorporates the effects of fault detection, fault isolation, and system reconfiguration. The modeling method must allow appropriate FDIR states to be defined and must be able to accommodate the differences in magnitude and statistical distribution between the failure occurrence rates and the FDIR rates.

The types of tools necessary to support this level of modeling are listed in Table 4.1.

With the more detailed models that can be constructed at this stage, the system engineer can make more accurate estimates of system performance and evaluate the functionality of the design. He can investigate different scheduling and allocation strategies and can evaluate the speedup in application compute time achieved by different parallel decompositions on different architectures. He can evaluate the sensitivity of system performance to architectural parameters, such as processor speed, communication bandwidth, and number of processors, and to the allocation of application processing and communication workloads to architectural components. He can also identify problem areas in particular designs.

At this stage in the design, when the reliability models are necessarily based on as-

Table 4.1. Tools for Initial Design Phase

Tool Used	Type	Purpose
Teamwork	CASE	Algorithm description
ADAS/EDIGRAF	Attribute model creation	Parameterized, hierarchical model creation
ADAS/ADLEVAL	Parameterized model definition	Algorithm description, function library
ADAS/ADLEVAL	Static analysis	Instantiated models
ADAS/GPSIM	Performance modeling via data flow and resource allocation	Performance estimates
ADAS/CSIM	Functional modeling and simulation	Examine functionality of design
Instrumented Code	Measurement	Provide performance data for refining performance models
ASSIST/SURE	Parameterized attribute model creation/reliability modeling	Probability of system failure with respect to mission time

sumptions regarding the performance and effectiveness of the FDIR mechanisms and the expected failure modes, the models can be used to evaluate the sensitivity of system reliability over the assumed range of FDIR performance and effectiveness. These analyses can establish bounds on the performance and effectiveness that the fault tolerance mechanisms must attain to achieve particular levels of reliability. These bounds provide a basis for conducting trade-offs of reliability gains from increased coverage of faults versus the performance cost of providing that coverage.

At the end of the initial design phase, the system engineer will be able to select a candidate design. He will have mid-level models of system components, including models that capture functionality. As the design progresses, these models can be refined with measured data and expanded to capture system behavior in faulted as well as fault-free operation.

4.2. Initial Design Case Studies

During Phase I, performance modeling case studies were conducted to explore issues of model construction, fidelity, and use at this level. These case studies illustrated the use of models to support the analysis of load balancing, resource scheduling and contention, deadlock prevention, and the performance of fault tolerance mechanisms.

More detailed models of intercomputer communications were explored for evaluating fault tolerance and task distribution mechanisms, and functional models were created to model shared memory, task-to-resource mapping, and scheduling mechanisms. A case study also looked at the use of measurement to support model development and validation.

The reliability analysis case studies in Phase I centered on the construction of a model of an FTPP cluster. The number of system components and the dependencies between failures of the network and processor components of the cluster precluded creating an exact model of the system, so the approach was taken of creating approximate, bounding models based on the underlying triad structure of the cluster configuration chosen for study.

The Phase I initial design studies are summarized in Table 4.2 and documented in [2].

Table 4.2. Summary of Phase I Case Studies for Initial Design Phase

Modeling Area	Analysis Area	Algorithm	Architecture
Performance simulation	Interprocessor communications	WTA/TS	FTPP
Functional simulation	Shared memory	WTA/TS	Multimax
	Reconfiguration algorithms		
Measurement	Model development and validation	WAUCTION	
	Mapping, Scheduling, communications, and resource contention	WTA/TS	Hypercube
Fault Tolerance Performance	Communications	WTA/TS	Multimax
	Reconfiguration algorithms		FTPP
Reliability Analysis	Model construction and reduction		FTPP

As a result of comparing modeling results with engineering model measurements for the WAUCTION algorithm in Phase I, an investigation was undertaken in Phase II to study the use of stochastic attributes to create an improved workload characterization. The description of the Weapon-to-Target Assignment/Target Sequencing (WTA/TS) algorithm was also refined in Phase II, prompting another iteration of performance analysis of WTA/TS on the Fault-Tolerant Parallel Processor (FTPP). The FTPP reliability model constructed in Phase I was extended to include spares and transient failures in Phase II and a range of parametric reliability analyses were conducted. A phased-mission model was also constructed and analyzed.

4.2.1. Improved Workload Characterization

4.2.1.1. Background

During Phase I of this effort, a model for the workload of WAUCTION-ASSIGNMENT, an algorithm which assigns weapon resources to specific targets, was developed. The model structure and parameters were based on an analysis of a program which implemented the algorithm. The model parameters associated with data-dependent loops in the program were estimated based on measurements obtained from instrumenting the implementation of the algorithm. Figure 4.3 is a procedure-call diagram of WAUCTION-ASSIGNMENT and indicates the data-dependent iteration parameters of interest. Average values or linear functions of the complexity variables of targets (T) or weapons (W) were used for the data-dependent loop parameters.

The performance predicted from the model was compared to the actual performance as measured from executions of the algorithm implementation. Figures 4.4 and 4.5 illustrate these results. The predicted results differed from the measured results in several respects. The measured workload as a function of the number of weapons did not increase as much as the predicted workload. While the measured and predicted workload as a function of the number of targets increased in similar ways, the measured workload showed substantial increases or peaks at certain values of weapons and targets. The increased workload indicated by these peaks was associated with the algorithm having greater difficulty in obtaining an optimum assignment for those values T and W. Peaks occurred when the number of targets (T) was about 1.6 times the number of weapon projectiles, where the number of projectiles is 5 times the number of weapons. Examination of measured data-dependent loop iterations within the algorithm indicated that certain of these iteration counts showed similar peaking behavior for $5 \times W \leq T \leq 1.6 \times 5 \times W$. It was concluded that WAUCTION-ASSIGNMENT workload could not be well characterized by simple monotonic functions of complexity variables such as N, N^2 or $\log N$. These results indicated a need to examine methods to improve the WAUCTION-ASSIGNMENT workload model.

4.2.1.2. Description

The objectives of the Phase II performance modeling associated with WAUCTION-ASSIGNMENT were to examine the extent to which stochastic models were appropriate for the WAUCTION-ASSIGNMENT workload model and to improve the workload model fidelity with respect to results obtained in Phase I.

To assess the value of stochastic models, the stochastic nature of the workload was examined. The instrumented implementation of WAUCTION-ASSIGNMENT was

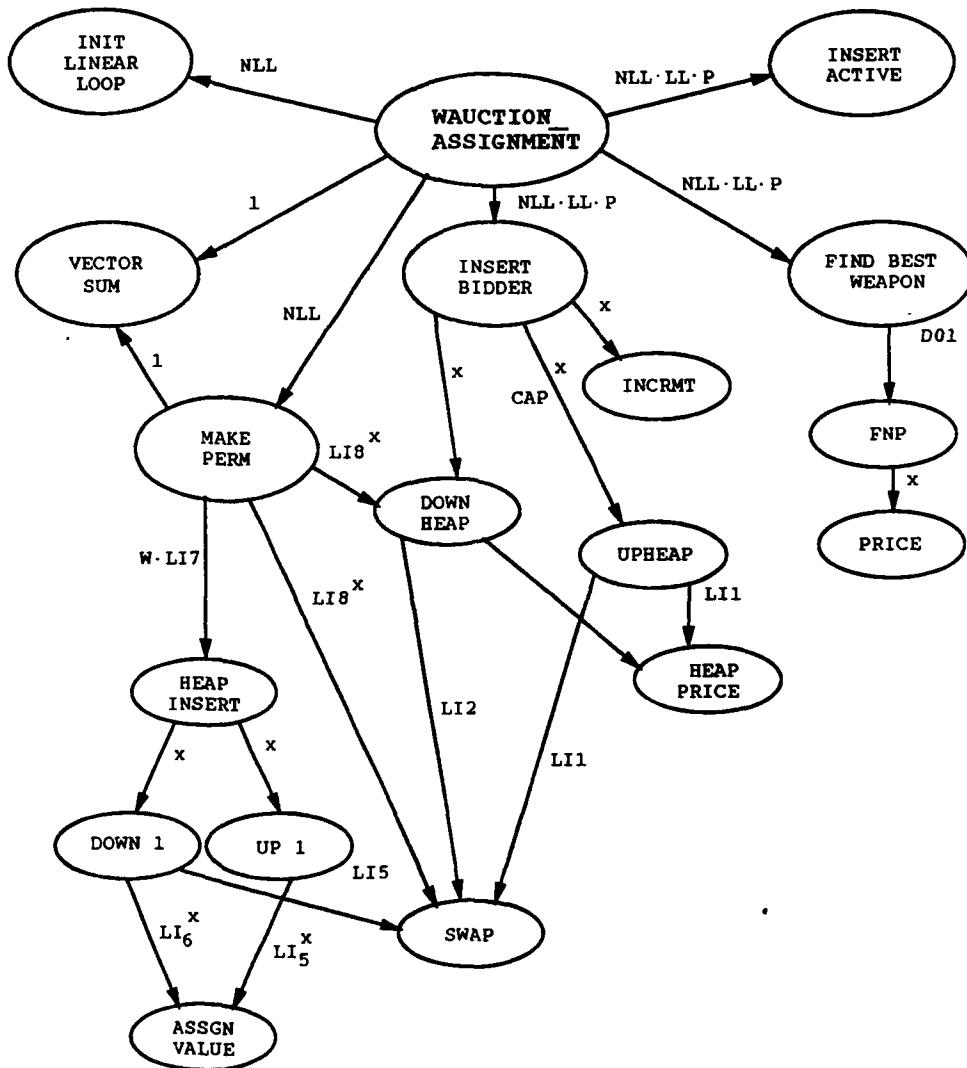


Figure 4.3. WAUCTION-ASSIGNMENT Procedure Call Diagram

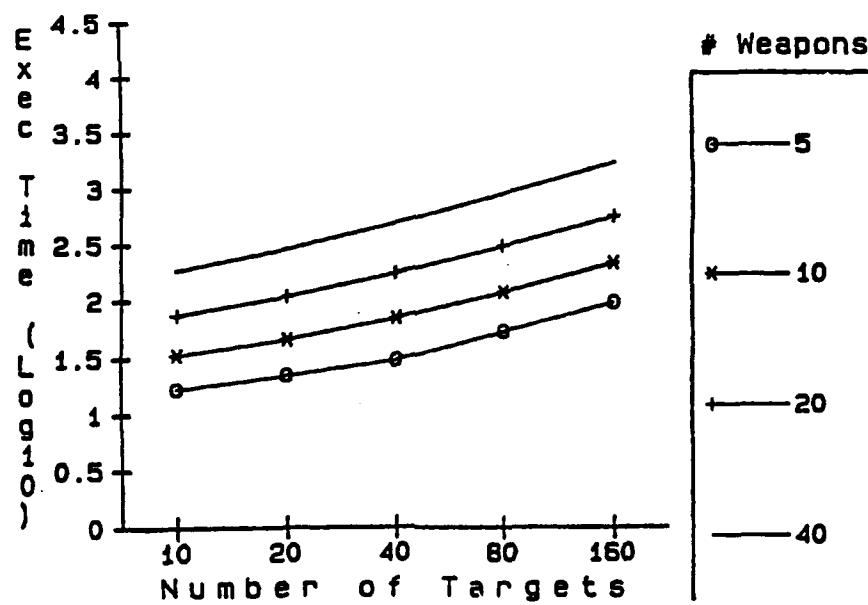


Figure 4.4. Computational Model Performance Results

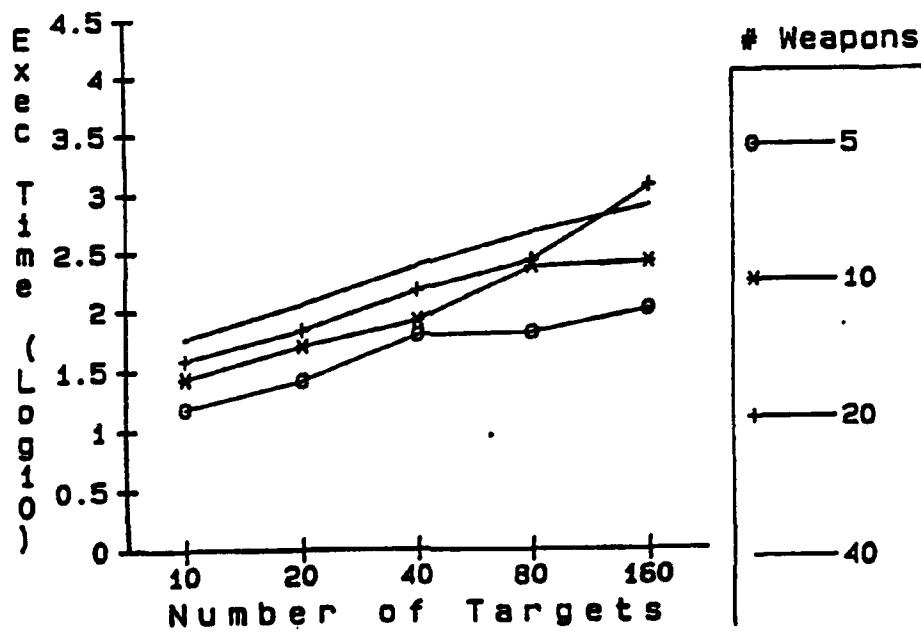


Figure 4.5. WAUCTION-ASSIGNMENT (Engineering Model) Measured Performance Results

exercised using randomly generated target kill probability arrays as input. For each combination of values for weapons and targets, 30 sample target kill probability arrays were tested. The number of weapons ranged from 5 to 40, and the number of targets ranged from 10 to 160. A total of 32 combinations of weapons and targets were tested. Measurements of workload and iteration parameters were collected and the minimum, maximum, average, and standard deviation was calculated for each set of parameter samples.

To improve the fidelity of the Phase I workload model, the use of more complex functions to describe the behavior of data-dependent loops was examined. Functions were derived to express the iterative behavior in terms of the number of available weapons and the number of expected targets. These functions were empirically fitted to the data from the measurements of workload and iteration parameters resulting from the study of the algorithm's stochastic behavior. These functions replaced the simple average value and linear, monotonic functions in the attribute definitions of the WAUCTION_ASSIGNMENT workload model. The model structure remained the same. Table 4.3 lists the attribute definitions used in the Phase I and Phase II models for the data-dependent loops in the algorithm.

4.2.1.3. Results

Examination of the sample iteration parameters indicated that the minimum and maximum sample values for a given target and weapon combination were, at worst, within 10% of the average value. Generally, less than 5% variation was observed. Since model-based workload estimates departed from measured workload values by greater than 10%, it was decided that a stochastic model for the workload was not appropriate.

The workload predicted using the Phase II functions for iteration parameters was more accurate than that from Phase I for target values between 10 and 40. For weapon values between 5 and 20 and target values between 40 and 160, the Phase II model still departed significantly from the measured values. The peaking associated with the product of the non-linear loop function, the linear loop function, the T/W pairing function, and the DO1 function has the greatest impact on the workload model. The functional descriptions of the iteration parameters of these loops were not sufficient for the model to track the peaking associated with the measured workload. The Phase II model results were, however, generally more accurate than those from the Phase I model.

Table 4.3. Phase I and II WAUCTION-ASSIGNMENT Iterative Loop Behavior Values

I.D.	Description	Procedure	Model	
			Phase I	Phase II
NLL	NONLINEAR LOOP	MAIN	10	$\frac{12 \left[1 + \left(\frac{T}{8W} \right)^{\frac{3}{2}} \right]}{1 + 0.2ABS \left[\frac{1}{4} \left(\frac{T}{W} \right)^2 - 1 \right]}$
LL	LINEAR LOOP	MAIN	2	$\frac{11W \left(\frac{T}{5W} \right)^{\frac{3}{2}}}{16(1 + .95ABS \left[\left(\frac{T}{5W} \right)^2 - 1 \right])} + .5$
P	T/W PAIRING	MAIN	$0.22 * T$	$2 + .41T - 2.1 \times 10^{-4}W^2T$
L14	D01	FIND BEST WEAPON	$0.6 * W$	$0.6 * W$
L13	D02	FIND BEST WEAPON	0.3	0.3
L18	M	MAKE PERM	$0.18 * T$	$0.2 * T$
L17	J	MAKE PERM	2	1
L16	DOWN 1	DOWNHEAP 1	1	$0.02 * T$
L15	UP 1	UPHEAP 1	0.3	$0.005 * T$

4.2.1.4. Conclusions

The variations in WAUCTION-ASSIGNMENT workload between sample target kill probability arrays tested in this case study were not significant enough to warrant developing a stochastic model. The use of more complex functions for data-dependent iteration parameters resulted in improved workload predictions. Additional changes to the iteration functions to improve predictions in the peaking region was not attempted. However, such modifications are straightforward and would lead to an even better model.

4.2.2. High-Level Performance Assessment for Parallel Decomposition

4.2.2.1. Background

As described in Section 3.2.1, the Phase I description of the Weapon-to-Target Assignment/Target Sequencing (WTA/TS) algorithm was revised to refine the analysis of its constituent functions. Using this description and a multicluster FTPP architecture, a study was conducted to determine the impact of parallelism on system performance.

4.2.2.2. Description

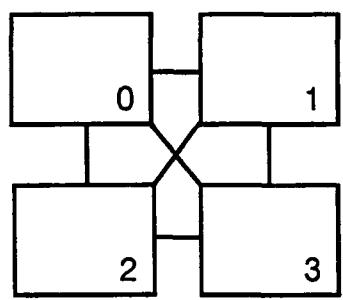
In order to model the system, assumptions had to be made about the algorithm, the architecture, and the mapping of the algorithm onto the architecture. The refined analysis of the WTA/TS algorithm took a closer look at the potential for parallel implementations of the algorithm's subfunctions. The model constructed for this case study captures the sequential-to-parallel mix of functions identified by this revised description.

Three cluster interconnectivity schemes were considered for analysis. These were fully interconnected clusters, linearly interconnected clusters, and clusters interconnected in a cube or ring. These are contained in Figures 4.6a, 4.6b, and 4.6c, respectively. The fully interconnected scheme was chosen as the "best case" example in that the maximum distance between triplexes is one cluster. The impact of communication distance will be discussed below in greater detail.

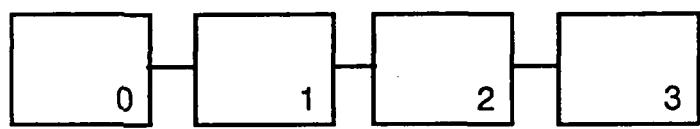
The WTA/TS algorithm was modeled as if executed on the fully interconnected FTPP architecture shown in Figure 4.7. Clusters are separated in this figure by dashed boxes. There are communication links between clusters indicated by solid lines. The intercluster communication channels connect input/output elements (IOEs) while network elements (NEs) provide intracluster communication channels.

Hardware speeds for the model were based on preliminary FTPP operating parameters. The processing element (PE) speed was set at 2,000,000 instructions per second or 2 MIPS. The NE bandwidth for this model was set at 500,000 bits per second (bps). The IOE bandwidth is 51.32 times slower than the NEs or ~9,000 bps.

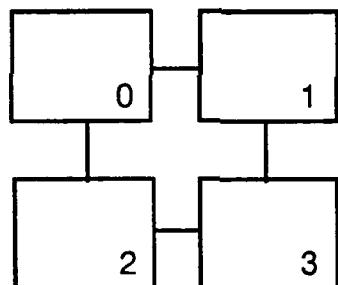
To demonstrate the impact of parallelism on the performance of the WTA/TS running on the four cluster FTPP, it was assumed that any one cluster would have four triplexes and an equal number of simplexes for spares. This assumption was consistent



(a)



(b)



(c)

Figure 4.6. Considered Cluster Interconnection Schemes

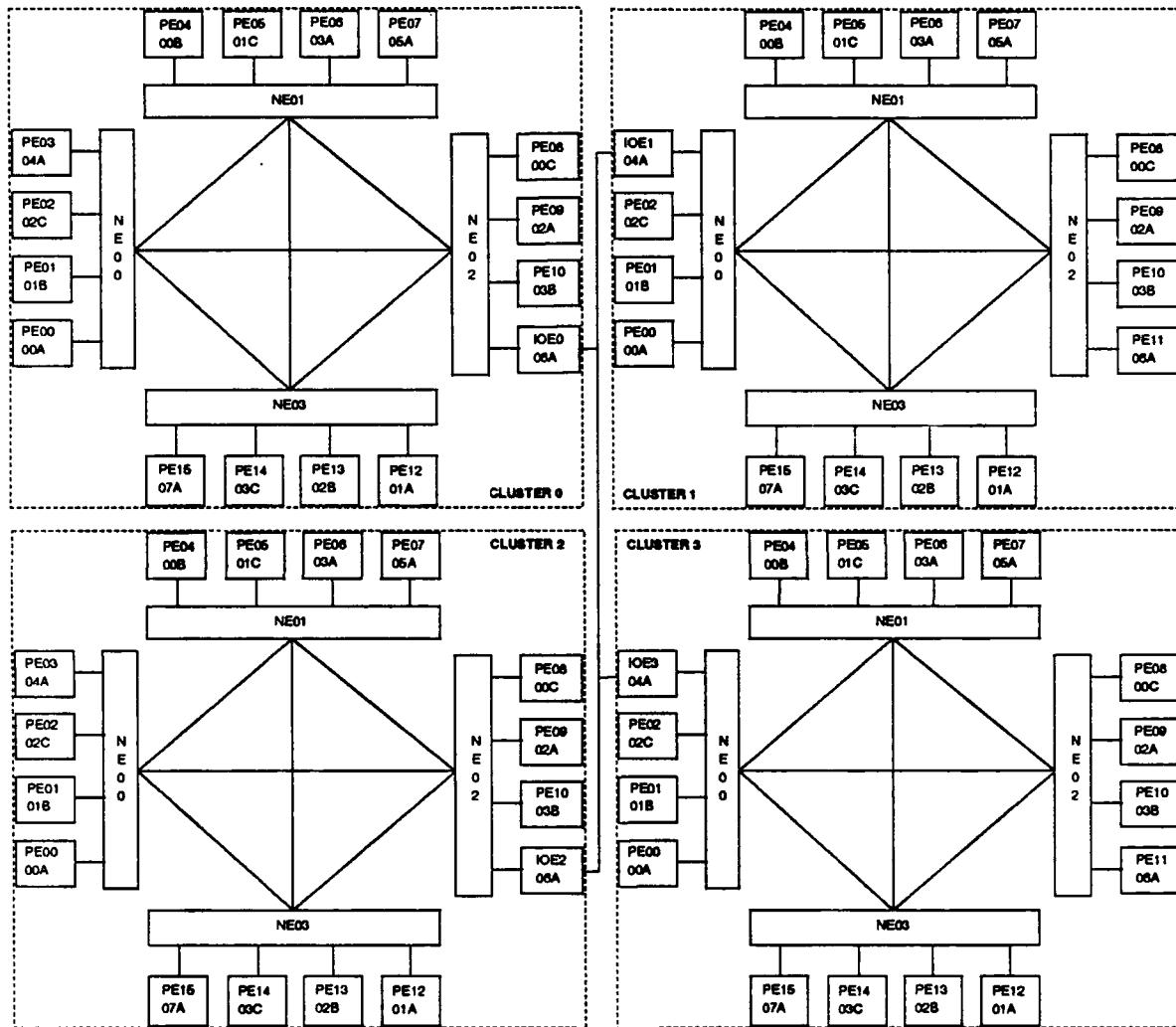


Figure 4.7. FTTP Configuration Studied

with many of the other Phase II studies involving the FTPP and limited the maximum degree of parallelism to four.

The designer of a multicluster FTPP must determine the appropriate action to take in the event of a triplex failure. It is conceivable that functions designated for execution on a faulty triplex would be moved and executed on the triplex of a neighboring cluster. Such a mapping was modeled to determine its impact on performance.

Table 4.4 depicts the algorithm-to-architecture mappings modeled. The hyphenated numbers in the matrix represent the number of parallel algorithm components running on each cluster. For example, "2-1-1-0" means that two of the four parallel components are running on two triplexes of cluster 0, while the other two components are running on one triplex of each of clusters 1 and 2. These mapping sequences were chosen for their clear representation of the impact on performance of parallelism across clusters.

Table 4.4. Explored Mappings

Number of Clusters	Parallelism			
	1	2	3	4
1	1-0-0-0	2-0-0-0	3-0-0-0	4-0-0-0
2	N/A	2-0-0-0	2-1-0-0	2-2-0-0
3	N/A	N/A	2-1-0-0	2-1-1-0
4	N/A	N/A	N/A	1-1-1-1

4.2.2.3. Results

Table 4.5 contains the time required to execute the mappings described in Table 4.4. Table 4.6 contains the dominant system component and its hardware component's utilization. "WAOpt" is the processor onto which the nonparallel Optimize subfunction of the Weapons Assignment function is mapped. By examining the row corresponding to execution on one cluster in Tables 4.5 and 4.6, it can be seen that when parallelism is increased, the required system execution time decreases, and the percentage of execution time devoted to components that can be executed in parallel decreases while the percentage of execution time devoted to components that cannot be executed in parallel increases.

The row representing execution on two clusters shows the performance impact of

Table 4.5. Time Required for the Execution of the Parallel WTA/TS Algorithm (in seconds)

Number of Clusters	Parallelism			
	1	2	3	4
1	69.332	61.030	59.070	58.090
2	N/A	61.030	414.855	594.436
3	N/A	N/A	414.855	594.436
4	N/A	N/A	N/A	861.152

Table 4.6. Dominant Algorithm Component (Utilization)

Number of Clusters	Parallelism			
	1	2	3	4
1	WAOpt(30%)	WAOpt(34%)	WAOpt(35%)	WAOpt(36%)
2	N/A	WAOpt(34%)	IOE(87%)	IOE(91%)
3	N/A	N/A	IOE(87%)	IOE(91%)
4	N/A	N/A	N/A	IOE(94%)

mapping parallel processes across clusters. For the two-cluster, two-parallel-process case, both processes are mapped to two triplexes on the same cluster. When the number of parallel processes is increased to three, the added process is mapped to a triplex on the neighboring cluster. The large increase in execution time between these two cases is directly attributable to the difference in speeds of the NE and the IOE. Since the IOE is 51 times slower than the NE, and one third of all parallel processes are utilizing the IOE communication path, it is no surprise that the IOE is the dominant system component and that the overall system performance is over six times slower than the former case. The three-cluster, three-parallel-process case is an identical mapping and produces the same results. This pattern of slowing down continues as half (two clusters, four parallel processes; three clusters, four parallel processes) and three-quarters (four clusters, four parallel processes) of the parallel processes are mapped to neighboring clusters.

The results of this modeling indicate that if the IOE is much slower than the NE communication channel; parallelism components should be executed on, at most, one cluster. However, if the IOE bandwidth were to increase to a level greater than that of the NE, cross-cluster execution of parallel function components would be a feasible option.

4.2.2.4. Conclusions

In an effort to satisfy system requirements, a designer must be able to perform system trade-off studies that exercise a large number of design options. In a parallel, fault-tolerant architecture, it is particularly necessary to be able to consider different workload-to-resource mappings to decide on parallel decompositions and recovery strategies. An effective design tool is needed that allows simulatable, hierarchical models to be created at the outset and refined throughout the design cycle. These models must permit different levels of abstraction to be captured so that specific system characteristics can be studied appropriate to the design phase, such as the concentration on communication mechanisms in this study.

4.2.3. Parametric and Phased-Mission Reliability Analysis

4.2.3.1. Background

In the initial design phase, reliability models can be used to evaluate the sensitivity of system reliability to proposed fault detection, isolation, and recovery (FDIR) mechanisms over ranges of performance and effectiveness assumed for those mechanisms. Critical parameters and assumptions can be identified for further analysis as the design develops and more accurate estimates of FDIR performance and effectiveness become available. As illustrated in Figure 2.10, the characteristics of the proposed space satellite mission and the diverse fault set it presents require a system with sophisticated fault tolerance mechanisms including system life extension mechanisms and complex fault recovery processes.

To develop a mission scenario representative of one that could be required for SDI, it was assumed that one computer system would support one platform in two mission phases: pre-engagement and engagement. The duration of the pre-engagement phase was assumed to be five years, that of the engagement one-half hour. It was further assumed that the system would support 256 parallel, reliable computations during the engagement phase with a maximum probability of system failure between 10^{-7} and 10^{-5} , and that the system would maintain adequate resources during the pre-engagement phase to meet the engagement requirements with a maximum probability of failure between 10^{-4} and 10^{-2} . Finally, it was assumed that permanent and transient faults would be tolerated, that spares would be used during pre-engagement, and that errors would be masked with no recovery during the engagement phase.

The Fault-Tolerant Parallel Processor (FTPP) is a high-performance, fault-tolerant architecture. It was chosen for the Phase II reliability analysis case studies to provide a current, complex problem for reliability modeling. Using the FTPP, parallel computations can be partitioned across clusters and across virtual group identifications (VIDs) within a cluster. Configuration tables resident in each processor identify group members. VIDs containing three or four processors are fault-masking groups (FMGs), where the occurrence of faults in an FMG causes no incorrect output. The VIDs are functionally synchronous and communicate via voted and source congruency messages. To further enhance fault tolerance by providing the basis for Byzantine Resilience, each member of an FMG resides on a separate network element (NE); a minimum of four NEs are involved in each message transmission. The scheduler is non-preemptive and tasks, which also communicate via messages, are scheduled once each iteration or when requested by a message. Faults are detected through syndrome information generated by the NEs and appended to each transmitted packet. A timekeeper function is allocated to one VID, but can be reassigned during reconfiguration. As soon as a vote detects an error, a fault diagnosis process identifies

the faulty component and a reconfiguration process removes it from active processing. So that transient faults do not cause premature depletion of resources, a faulty processor can be tested and restarted. Two reconfiguration strategies exist for the FTPP: total cluster reconfiguration and processor replacement. The total reconfiguration strategy takes into consideration the current system configuration and the requested system configuration to create and disband VIDs as necessary to achieve the requested configuration. Processor replacement reconfiguration begins by searching for a simplex VID in the correct position for inclusion in the VID undergoing reconfiguration. The faulty VID is then replaced with the selected simplex, and the faulty processor becomes a simplex VID that can resume in its initial state or enter a diagnostic phase.

Based on high-level estimates and other FTPP modeling results [29], and assuming triplex redundancy, an initial configuration of 768 active processors in 16 clusters was proposed for the case study. Each cluster would consist of 48 active processors (comprising 16 triads) and 152 spare processors. This gives 25 NEs per cluster, assuming that eight processors could be assigned per NE. A complete Markov model of one cluster in this configuration contains more than $2^{25} \cdot 2^{200}$ states. Sequence dependencies introduced by the distribution of the members of the triads over the network elements and strategies for recovery further complicate the model. As pointed out in [29], the creation and solution of a model for such a system relies on the aggregation and truncation of states in and paths through the model. In addition, the estimation of multiple-cluster reliability requires more sophisticated network reliability tools than are readily available.

For the goal of this case study, it is more useful to scale the proposed configuration down to a level where visibility into the model is possible and shorter model solution times allow a greater range of parameter values to be explored for each analysis. Therefore, the one-cluster FTPP configuration illustrated in Figure 4.8 was selected for this case study. This configuration consists of four triads distributed over four network elements, where PE_{ij} represents the j th channel of the i th triad. One spare processor element is connected to each network element, where S_k represents the k th spare. The network elements are fully connected within a cluster. The input/output processing elements (IOEs) would connect this cluster to other clusters. To accommodate the scaled-down configuration, the duration of the pre-engagement phase was assumed to be 10,000 hours rather than five years.

4.2.3.2. Description

The goal for this case study was to investigate reliability modeling issues through sample reliability evaluations of the selected FTPP configuration. The approach

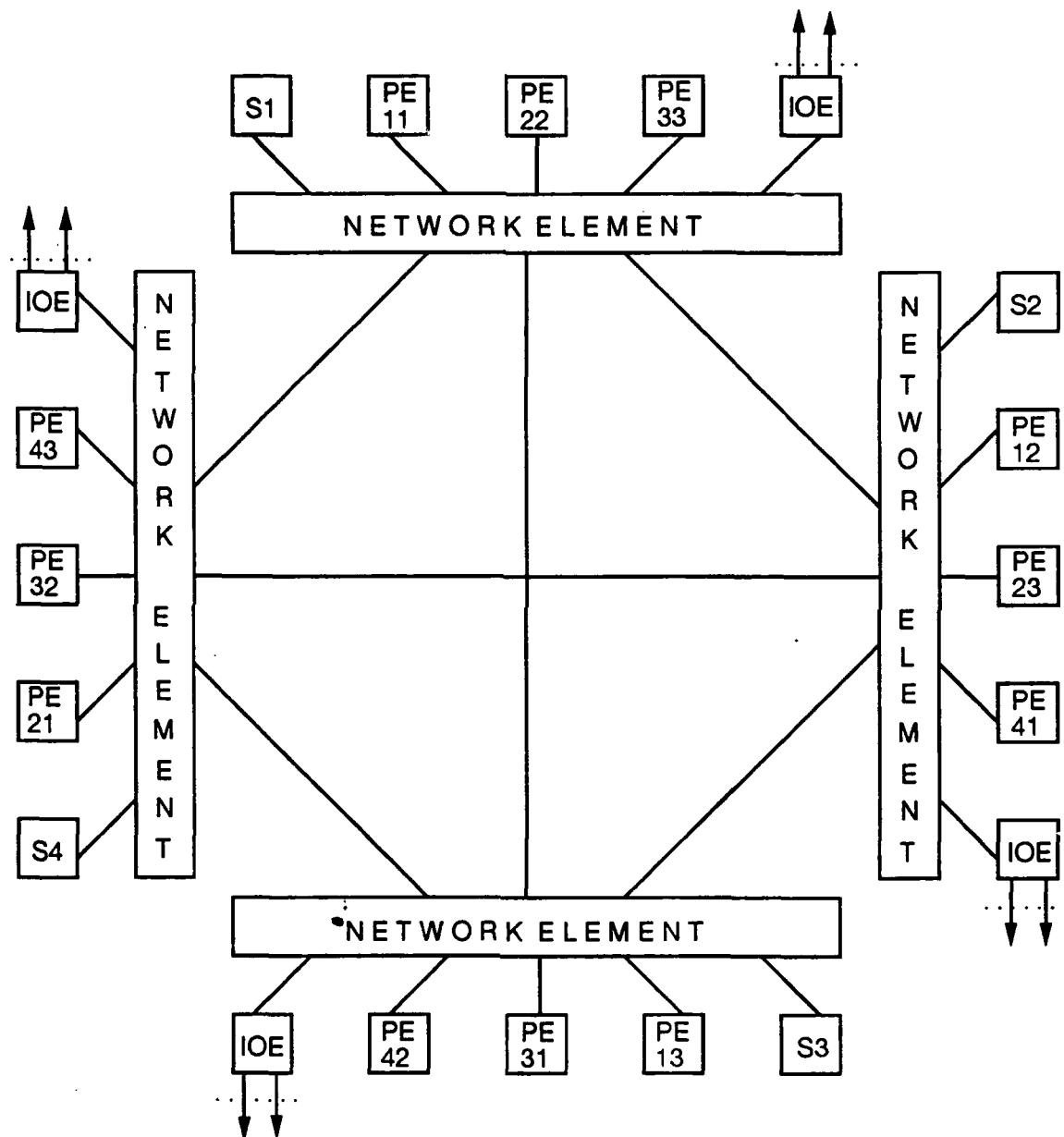


Figure 4.8. Sample FTPP Cluster Configuration

taken was to construct Markov models of the sample system that could be used to evaluate the reliability of the sample system for a phased mission representative of one that could be required for SDI. Through the use of ASSIST, parameterized models were created that could be instantiated and evaluated, using SURE and STEM, for a range of parameter values and system assumptions. From these analyses, critical parameters and assumptions could be identified.

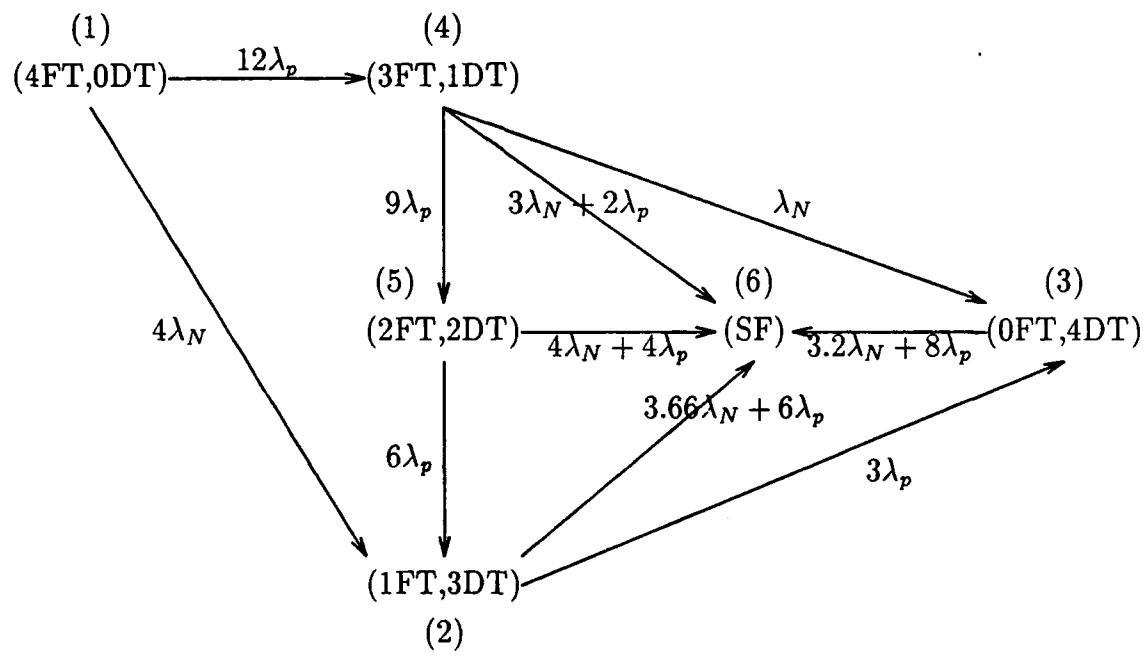
The FTPP strategy of immediate removal of a faulty processor regardless of the transient nature of the fault reduces its vulnerability to a second fault while the transient is present. This strategy introduces a subtle modeling issue since the fault handling models of some tools were not designed to accommodate this reduced vulnerability. The division of the mission into two phases requires the construction of two models, the identification of mission-end operational states in the first phase model with states in the second phase, the ability to compute occupancy probabilities for the mission-end operational states, and the ability to set initial occupancy probabilities for states in the second phase based on the first-phase mission-end occupancy probabilities.

During Phase I of DAHPHRS, an approximate, conservative model (see Figure 4.9) of a similar FTPP configuration was constructed by comparison to and aggregation from other bounding models. This model aggregates the occurrence of faults on specific triads according to the total number of failures that have occurred and the distributions of those failures across triads. It captures interdependencies between the triads and the network elements (NEs); however, it overestimated the loss of processors (PEs) by aggregating certain combinations of network element and processor failures in worst-case states. For this study, a generalized ASSIST model was created from the Phase I model that could handle variable numbers of NEs and of PEs per NE and could model recovery and reconfiguration behavior. One of the essential difficulties in creating such a model is representing in the general case which NE failures lead to system failure in the presence of previous failures, since this is entirely dependent upon the exact configuration at the time of the failure.

The generalized model was constructed iteratively as a means of verifying its validity. The first step was the creation of an ASSIST model that duplicated the Phase I model. Next, the model was modified to include a more general state space definition, but with transitions that were valid for the specific four-triad, four-NE case. Finally, the model was modified to generalize the transition definitions. At each step of model construction, the model and its reliability predictions were compared with those from the previous step.

A number of assumptions were made to enable the specification of dependencies between NE and PE failures:

- An NE failure causes the initial number of active PEs per NE to fail. An NE



$(iFT, jDT) \equiv (i$ Full Triads, j Degraded Triads)
6 States, 10 Transitions

• Figure 4.9. Phase I Model of 4×4 FTPP Cluster Configuration

failure also reduces the number of full triads by the same number. This is a conservative estimate, because the initial number of active PEs per NE decreases as PEs fail.

- An NE failure causes the initial number of spare PEs per NE to fail. This is also a conservative assumption.
- If the number of degraded triads, i.e., a triad which has one failed member, is greater than half the number of NEs, any network element failure will cause system failure. This is due to the fact that an additional PE failure on a degraded triad will cause system failure. If there are a sufficient number of degraded triads, the PEs failing as a result of an NE failure will most likely be members of an already degraded triad, causing system failure. A conservative selection of one-half the number of NEs is based on the enumeration of several cases.

Finally, it was assumed, based on the initial description of the FTPP reconfiguration strategy, that if a spare was available, the system could be reconfigured to replace any faulty processor. A composite diagram of the resulting model is illustrated in Figure 4.10. The states in this model represent the number of full triads, degraded triads, and spares. The transitions are due to combinations of PE and NE failures. System failure is defined to be the loss of any of the required number of triads. This model can handle reconfiguration with spares and variable numbers of triads, NEs, and PEs per NE. The model handles transient failures, including a waiting time for diagnostic tests of the processor and a possible return to the spare processor pool. It also models hot or cold spare processor failures. NE failures are modeled, including subsequent processor failures for varying numbers of existing processor failures. The loss of Byzantine Resilience due to insufficient numbers of active NEs is not modeled. For the configuration with four triads and four spares connected to four NEs, this model generated 97 states and 486 transitions.

4.2.3.3. Results

4.2.3.3.1. Sensitivity Analyses

The characteristics of the mission and the modeled architecture result in a number of parameters that determine overall system reliability. A number of analyses were conducted of the type necessary to evaluate the sensitivity of system reliability to variations in model parameters such as failure rates and recovery rates. The parameters of the FTPP model for which these analyses were conducted included permanent and transient processor failure rates, permanent and transient recovery rates, and

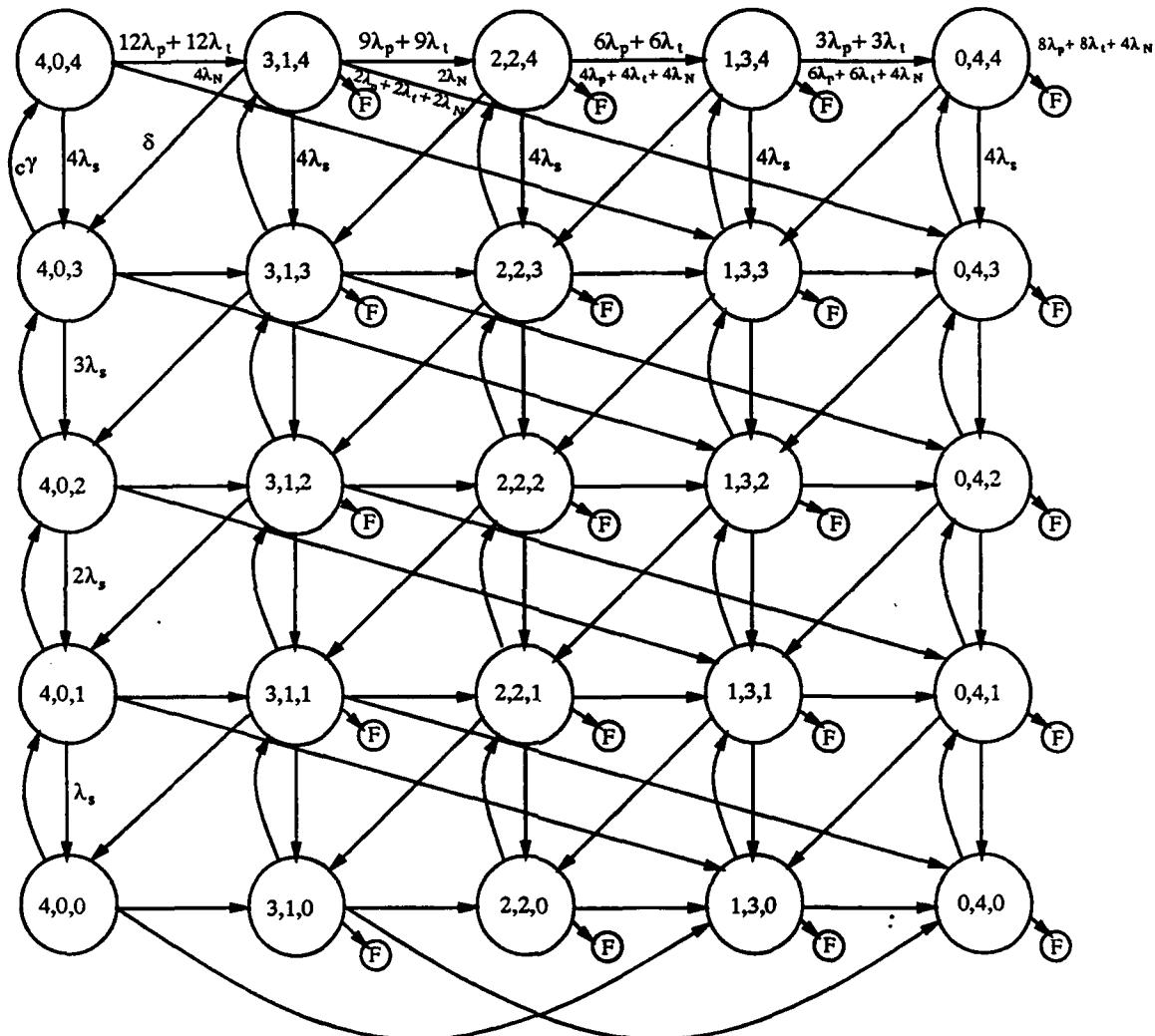


Figure 4.10. FTPP Markov Model

network element failure rates. To isolate the effects of a particular parameter, the effects of other parameters were nullified during analysis.

The permanent processor failure rate, λ_p , varies with the complexity of the device, the technology of implementation, and the environment in which it operates. To assess the sensitivity of system reliability to variations in permanent processor failure rate, λ_p was varied over the range 5×10^{-4} to 10^{-5} . No transient processor or network element failures were considered. A "standard" recovery rate, δ , of one recovery per second was assumed. Figure 4.11 illustrates the results of this group of analyses for mission times from 1 to 100,000 hours. It can be seen from this Figure that a permanent processor failure rate not greater than 10^{-5} would be required to meet the 10^{-4} to 10^{-2} requirements of a 10,000-hour mission.

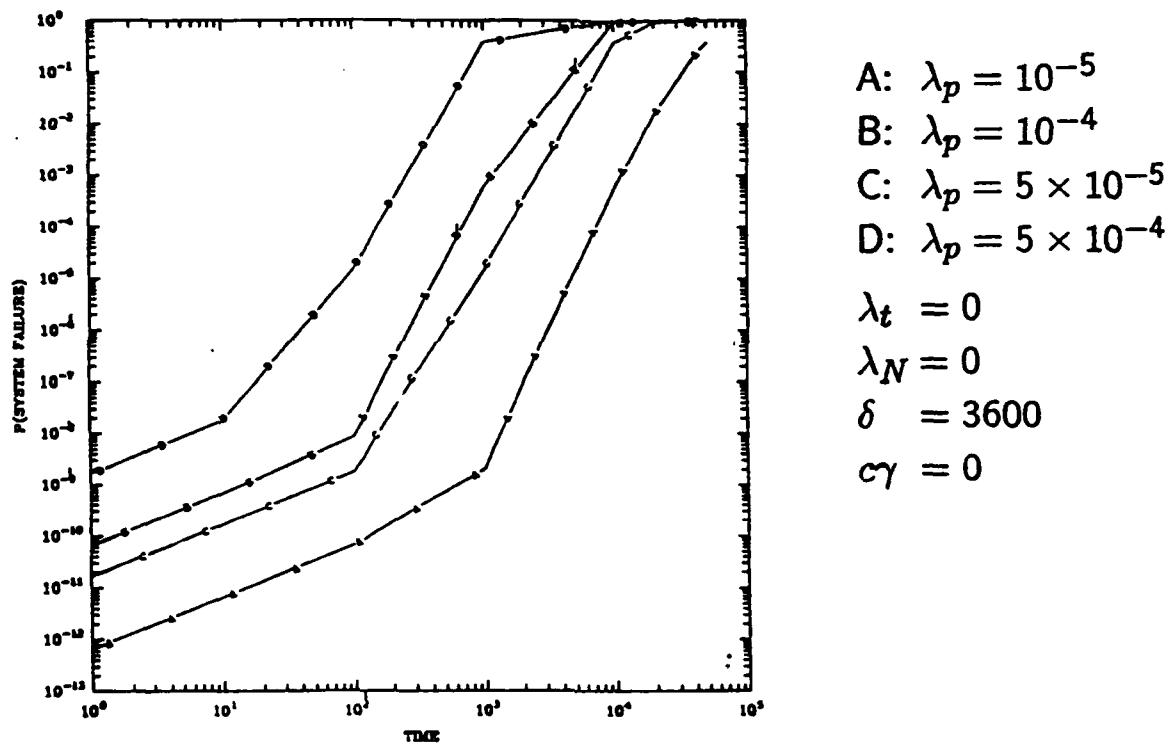


Figure 4.11. Results of Permanent Processor Failure Rate Analysis

After estimating bounds for processor failure rates assuming a fixed recovery rate δ , a group of analyses was conducted to determine the effect of variations in δ . For these analyses, λ_p was set to 10^{-5} , the upper bound determined from the previous analyses. No transient or network element failures were considered. The rates selected for δ ranged between one recovery per hour and one per second. The results are illustrated in Figure 4.12.

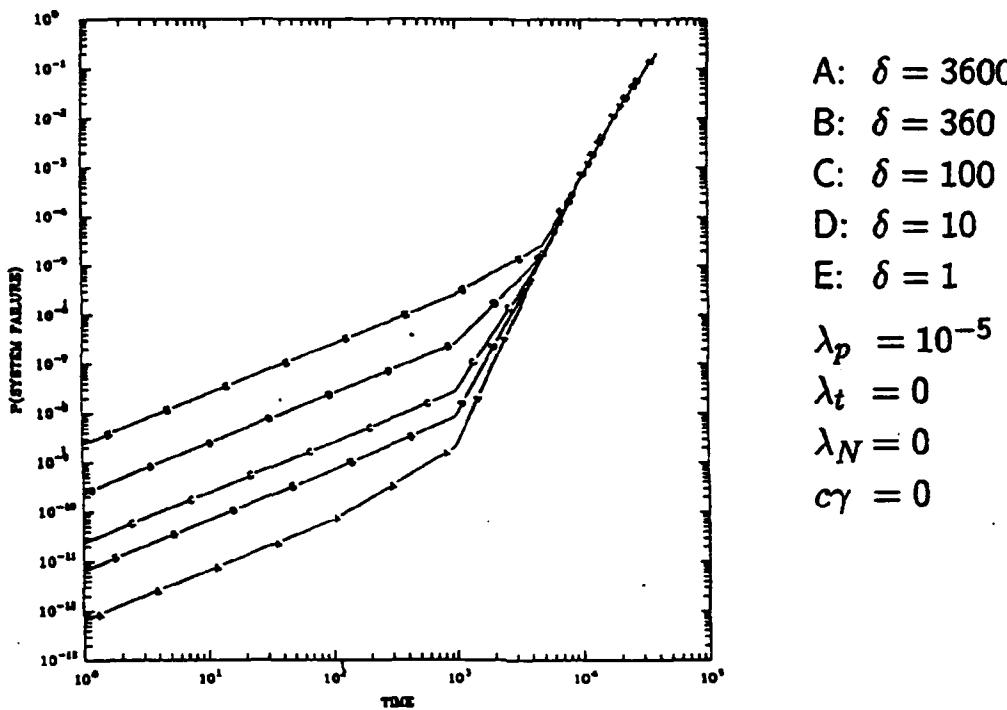


Figure 4.12. Results of Processor Recovery Rate Analysis

As expected, the faster rates result in significant increases in system reliability in the early portions of the 10,000-hour mission, but the effects of differences in recovery rate decrease over time. For each of the selected rates, the effects started to vanish around 1,000 hours; by about 5,000 hours there is no significant difference in system reliability for different recovery rates. This is due to the fact that exhaustion of parts starts to dominate the system reliability.

The next group of analyses looked at the transient processor failure rate, λ_t . No permanent processor or network element failures were considered. Since transient failures are often assumed to occur ten times more often than permanent failures, and since the permanent failure rate analyses determined a maximum permanent failure rate of 10^{-5} , values for λ_t of 10^{-4} and 5×10^{-4} were selected for these analyses. Obviously, the success at identifying transients and restoring the processors to an active state will determine whether or not the reliability requirements are met under these failure assumptions. Therefore a parameter $c\gamma$ consisting of the probability that a fault is transient (c) and the rate at which it is recognized as such and the processor restored (γ) was included in the model for transient recovery transitions. For these analyses, $c\gamma$ was varied over a range of 100 to .01 per hour. Also, since the FTPP FDIR strategy is to remove any faulty processor at once and replace it with a spare before test and/or restart and return of the processor to the spare pool, the recovery

rate δ is required and was assumed to be one per second. The results of these analyses are illustrated in Figure 4.13. As can be seen from this Figure, to attain the 10^{-4}

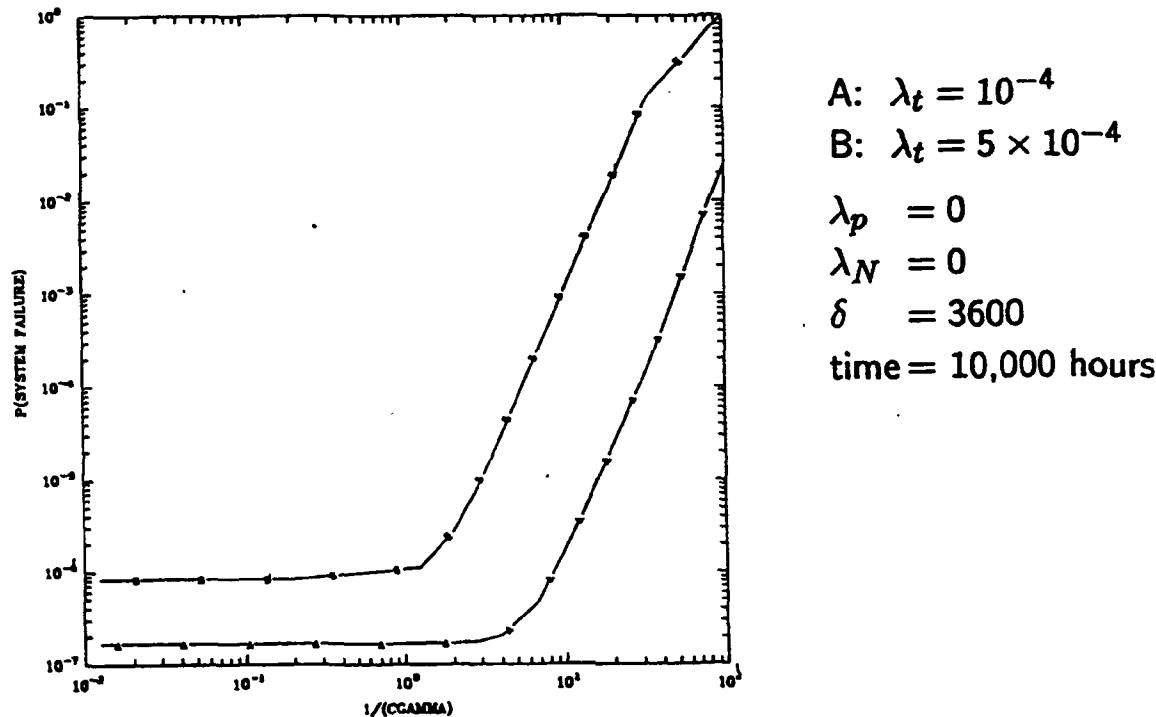


Figure 4.13. Results of Transient Processor Failure and Recovery Rate Analysis

to 10^{-2} requirements for a 10,000-hour mission, the minimum value for $c\gamma$ must be between .03 and .01 for $\lambda_t = 10^{-4}$ and between 2 and .05 for $\lambda_t = 5 \times 10^{-4}$.

Having looked at the effects of permanent and transient processor faults and recovery, the next group of sensitivity analyses looked at the effects of different NE failure rates, λ_N . For this set of analyses, no transient failures were considered. The values selected for λ_N were 10^{-5} , the same as the processor failure rate; 10^{-6} , ten times less often than processor failures; and 0, assuming perfect operation. The results are illustrated in Figure 4.14. As can be seen from this figure, NE failures have a significant impact on system reliability, and their rate of occurrence has to be significantly lower than that of the PEs for the 10^{-4} to 10^{-2} requirements to be met. This is due to the NE being a resource that is shared across multiple triads. It is assumed that since it is a much simpler device than a PE, a lower failure rate can be attained. However, it would clearly be necessary to consider during design whether or not the functionality required of the NE to support the full requirements of the fault detection and isolation mechanisms and the task communication mechanisms would require a more complex device than initially planned.

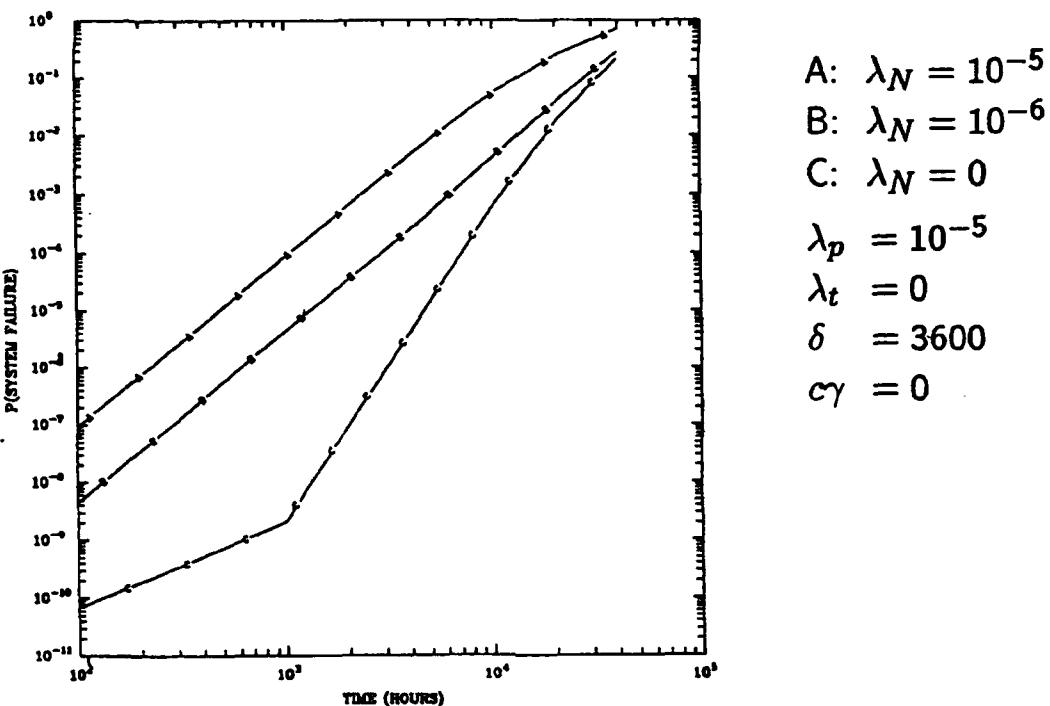
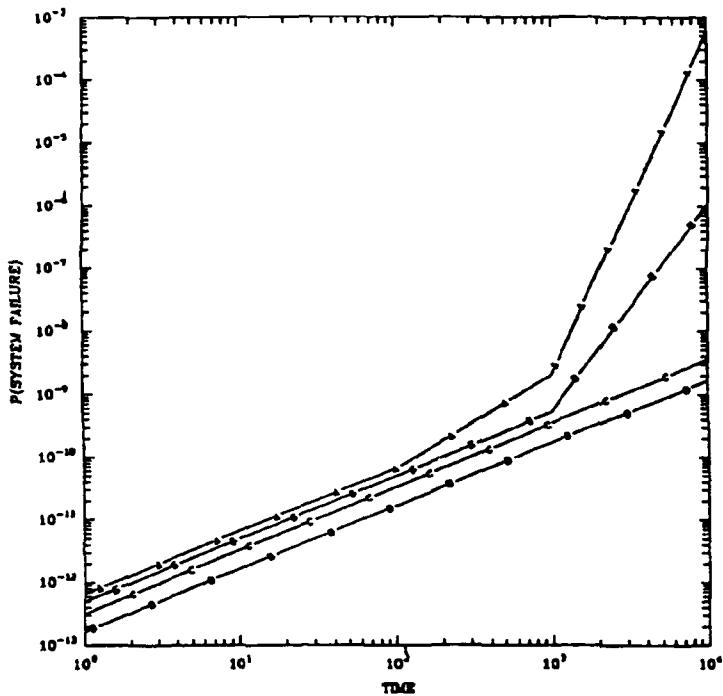


Figure 4.14. Results of Network Element Failure Rate Analysis

4.2.3.3.2. Architecture Configuration Analysis

The second type of analyses appropriate to the initial design modeling phase that were conducted in this case study are analyses that would be useful in determining trade-offs of initial and degraded availability of resources for computational performance against minimum assignment of resources to attain reliability requirements. For this case study, an example of reliability analysis of different configurations of available hardware resources in the selected four NE, four PE per NE architecture, was performed. For this set of analyses, λ_p was set to 10^{-5} and δ to one per second. No transient or network element failures were considered. The results for configurations of from one to four triads are illustrated in Figure 4.15. In the first 100 hours, the relative increase in system reliability from fewer required triads is constant. At about 1,000 hours, the reliability of the four- and three-triad configurations rapidly decreases. While the four-triad configuration meets the 10^{-4} to 10^{-2} requirements for 10,000 hours, the remaining configurations well exceed those requirements due to the increased sparing capability. Thus, substantial improvement in reliability would be gained by not utilizing the maximum parallel performance available in the architecture. Looking at the results of a performance analysis of a mission planning algorithm on a four-NE, four-PE-per-NE FTPP summarized in Table 4.5, the time to compute the algorithm using three processing sites is about 2% longer than using



- A: 4 triads, 4 spares
- B: 3 triads, 7 spares
- C: 2 triads, 10 spares
- D: 1 triad, 13 spares

$$\begin{aligned}
 \lambda_p &= 10^{-5} \\
 \delta &= 3600 \\
 \lambda_t &= 0 \\
 \lambda_N &= 0 \\
 c\gamma &= 0
 \end{aligned}$$

Figure 4.15. Results of Variable Number of Triads Analysis

four sites. The results of both of these analyses would be useful in determining the final configuration of the architecture.

4.2.3.3.3. Phased Mission Analysis

The final analyses conducted in this case study were phased-mission analyses. Phased mission analysis for the pre-engagement, engagement scenario was conducted by solving a pre-engagement phase model and using the resulting state occupancy probabilities to initialize an engagement phase model. This method allows the computation of the conditional probability of success for the engagement phase, given that the pre-engagement phase was successful, but requires a tool that can report occupancy probabilities of operational as well as failed states. For the pre-engagement phase scenario, the selected FTPP configuration relies on spares to attain the reliability requirements. During this phase, it relies on Byzantine Resilient communications and voting to mask errors. The complete model described in Section 4.2.3.2 was required. For this analysis, a configuration of four triads distributed over four NEs with a permanent processor failure rate of 10^{-5} for both active and spare processors and an NE failure rate of 10^{-6} were selected based on the results of the sensitivity analyses. The processor recovery rate, δ , of one per second used in those analyses

was also used in these analyses. Since the capability of achieving the required times for transient recognition and recovery that the sensitivity analyses identified did not seem to be a critical issue and since the inclusion of transients slowed down the execution of the solution for long mission times by introducing path loops through the model, transient failures were not considered for the pre-engagement phase.

For the engagement phase scenario, the selected FTPP configuration relies on the same mechanisms as in the pre-engagement phase to mask the effects of any single fault. However, no recovery or reconfiguration was attempted. The model illustrated in Figure 4.16 was required for this scenario. It was assumed that during the half-hour

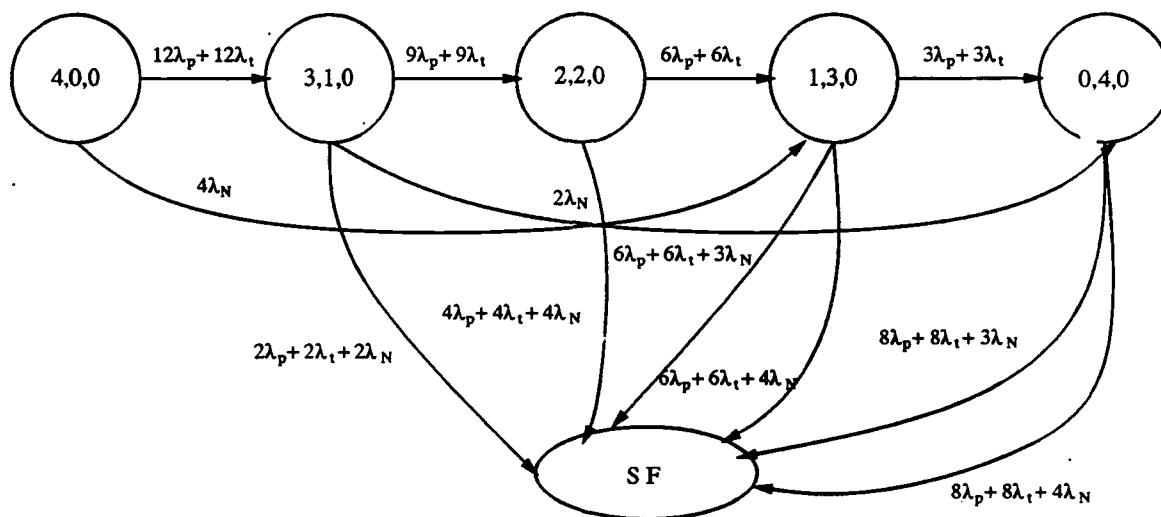


Figure 4.16. Engagement Phase Model

mission time, the triplex redundancy of each processing site and the Byzantine Resilient communications provided by the network elements would be sufficient to mask faults via voting. The maximum probability of system failure during the engagement phase should be between 10^{-7} and 10^{-5} .

The pre-engagement state occupancy probabilities were computed and the relevant operational states identified. Since the engagement phase could start at any state of the pre-engagement model, each state of the engagement model had to be initialized to the sum of the occupancy probabilities of the appropriate pre-engagement states. If the states of the pre-engagement model in Figure 4.10 are considered as a matrix, the states of the engagement model in Figure 4.16 correspond to the last row of that matrix. The initial occupancy probabilities of the engagement model states were computed by summing the ending occupancy probabilities of the states in the corresponding column of the pre-engagement model. Since the pre-engagement model, when generated, actually had additional dimensions due to the need to keep track of additional state discriminators, identifying and computing the occupancy

probabilities required the creation of a simple computer program to search the model solution output file for patterns of state vectors, sum up the occupancy probabilities of the matches for each pattern, and produce an output file of initial probabilities that could be included in the engagement model specification.

In addition to the phased mission scenario described above, an analysis was conducted assuming that recovery would be attempted during the half-hour engagement phase. For this analysis, the engagement phase model was identical to the pre-engagement phase model. The method of initializing the engagement phase model was the same.

The results of the phased mission analyses are summarized in Table 4.7. The param-

Table 4.7. Phased Mission Analysis Results

Mission Definition	Failure Rates	Recovery Rate	P(F)	P(S)
Pre-Engagement Phase (10,000 hours) P(SF) Req: 10^{-2} to 10^{-4}	$\lambda_P = 1 \times 10^{-5}$ $\lambda_S = 1 \times 10^{-5}$ $\lambda_T = 0$ $\lambda_N = 1 \times 10^{-6}$	$\delta = 3600$	3.98×10^{-3}	.996
Engagement Phase (.5 hour) P(SF) Req: 10^{-5} to 10^{-7}	$\lambda_P = 1 \times 10^{-5}$ $\lambda_S = 0$ $\lambda_T = 1 \times 10^{-4}$ $\lambda_N = 1 \times 10^{-6}$	No recovery	2.85×10^{-6}	.999997
	$\lambda_P = 1 \times 10^{-5}$ $\lambda_S = 1 \times 10^{-5}$ $\lambda_T = 1 \times 10^{-4}$ $\lambda_N = 1 \times 10^{-6}$	$\delta = 3600$ $c\gamma = .8$	3.8×10^{-7}	.9999996
TOTAL MISSION •	$\begin{aligned} P(F_M) &= 1 - P(S_P)P(S_E S_P) \\ &= 1 - (.996)(.999997) \\ &= 4.1 \times 10^{-3} \end{aligned}$			

eter values used for each of the models are listed along with the estimated probability of system failure, $P(F)$, and the resulting probability of success, $P(S)$, for each phase. The probability of system failure for the total mission, $P(F_M)$, is $1 - P(S_P) \cdot P(S_E | S_P)$, where $P(S_P)$ is the probability of success for the pre-engagement phase and $P(S_E | S_P)$ is the conditional probability of success for the engagement phase given that the pre-engagement phase was successful. As would be expected, the reliability of the engagement phase with recovery is significantly greater than without recovery, and total mission reliability, from the standpoint of the hardware architecture, is dominated by the reliability of the pre-engagement phase.

4.2.3.4. Conclusions

The FTPP reliability analysis case study demonstrated a set of analyses that can be used in the initial design phase to develop a candidate design and establish bounds on the performance and effectiveness that the proposed fault tolerance mechanisms must attain to achieve particular levels of reliability.

Three areas that present modeling problems have been highlighted in the development of models of the FTPP: parallelism, shared regions of connectivity, and communications. Larger, more complex models result from parallel architectures due to the increased numbers of system components and the increased interactions and dependencies between failures. The network element used in the FTPP is a shared region of connectivity between the processing elements of the architecture. As a shared resource, it introduces dependencies between failures. As a reconfigurable shared resource, the specific impact of its failure on the availability of attached processing elements is difficult to capture in a model. However, it is important that this impact be carefully gauged since the network element must be designed to be sufficiently simple to achieve a low enough failure rate to compensate for causing multiple losses while at the same time complex enough to handle the communications mechanisms required of it by fault detection and isolation processes and parallel task communication. The communication needs inherent in many distributed or parallel architectures result in large, complex networks that can not be adequately evaluated by current tools.

The creation and validation of the necessary models is complicated both by the need to include recovery states and transitions that are difficult to define and measure rates for and by the necessity of capturing all the relevant failure modes. The solution of these models for long mission times is made more difficult by mixed statistical distributions of varying orders of magnitude for the various transitions. Additionally, some of the useful truncation techniques for reliability modeling are not applicable when a significant contribution to system failure is made by states reached through multiple failure occurrences.

The aims of the modeling at this level of design require a large number of analyses across a wide range of system parameters. This requires a model whose attributes can be expressed parametrically and instantiated for each analysis. It also requires quick computation of model solutions. The volume of output from a large number of analyses also requires some automated mechanism of organizing that output and selecting the pertinent information for each analysis.

5. Design Refinement Modeling Phase

5.1. Description

Once the candidate design has been selected, the design refinement modeling phase is initiated. This phase corresponds with iterations through the Working Group design steps, illustrated in Figure 5.1, where the fault detection, isolation, and recovery

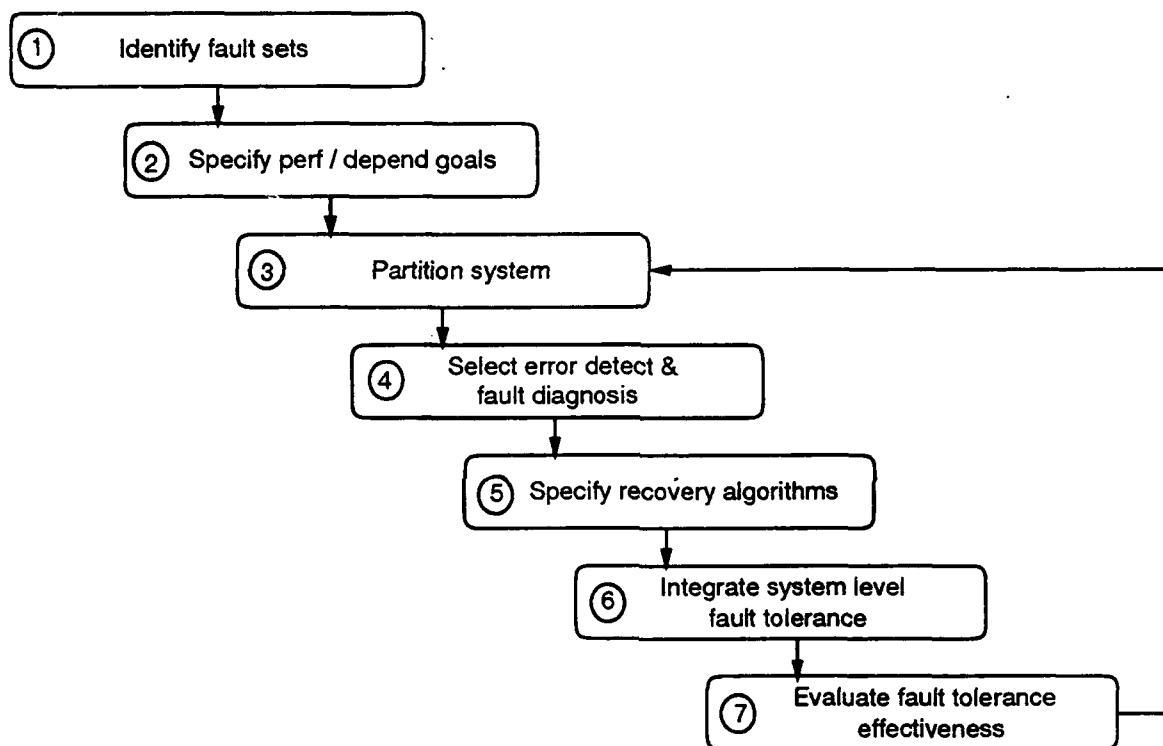


Figure 5.1. Design Refinement Relationship to Methodology

(FDIR) mechanisms are being developed in detail. The models and analyses in this phase focus on the fault tolerance mechanisms and their interaction with the application and operating system software.

The design refinement modeling process is illustrated in Figure 5.2. As indicated in this figure, the architecture and algorithm descriptions of the candidate design from the initial design phase are required along with descriptions of the operating system, the FDIR algorithms and hardware mechanisms, and the fault model to create the system behavioral model required for this phase. The performance models of these elements that have been developed during the course of the design phase must be

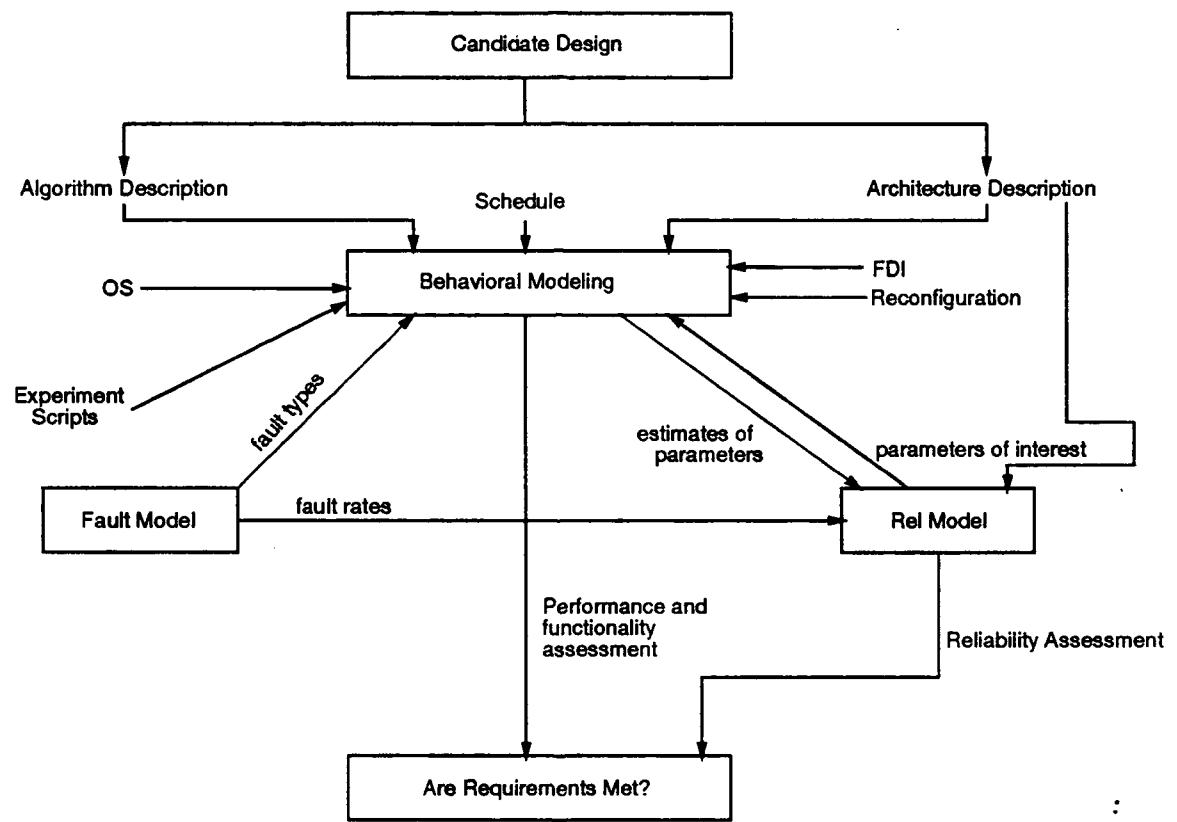


Figure 5.2. Design Refinement Phase Process Diagram

extended to functional models and integrated as illustrated in Figure 2.12 to capture system behavior. It is only through such an integrated model that the interactions between the FDIR mechanisms and the application and operating system functions can be studied.

Three distinct types of models are needed to support the desired evaluations: functional/behavioral, performance, and reliability. Functional/behavioral models can be used to assess the correct operation of targeted system elements under faulted and fault-free conditions to assess the effectiveness of fault isolation, error detection, fault diagnosis, error recovery, system recovery, and sparing strategies. Performance models can be used in conjunction with the functional/behavioral models to determine response times, resource utilization of the various fault tolerance strategies, and the system capacity in various degraded but operational states.

System reliability can be assessed by analytical reliability models that describe the behavior of the system in response to the occurrence of faults and that predict the probability that the system will be operating correctly at a given time. Markov and semi-Markov models have proved useful in assessing system reliability, but they rely on accurate estimates of fault arrival rates and distributions, on estimates of the time required to detect faults and isolate and recover from them, and on measures of the likelihood of these mechanisms succeeding. Currently, such FDIR parameters are estimated based on prior designs or measured in experiments on prototypes of the system. From analysis of an integrated fault-tolerance/performance model, estimates of these transition rates can be obtained for the current design and the target application before prototypes are constructed. Therefore, a more accurate assessment of overall system reliability can be made for design trade-offs.

The types of tools necessary to support this level of modeling are listed in Table 5.1.

In the design refinement phase, the focus can shift to a finer evaluation of a particular design, identifying design deficiencies and areas needing improvement, determining more realistic estimates of parameters such as recovery time or detection coverage, determining the behavior of fault tolerance mechanisms in the presence of faults, and finding and eliminating conceptual or requirements-level design errors in the fault tolerance mechanisms.

The behavioral model provides the higher level of model fidelity required by these activities. It integrates those models of system components that are necessary to simulate the actual processing of data at a functional level under both faulted and fault-free conditions. The design refinement analyses result in more accurate performance and reliability measures and a more detailed functional evaluation of the system design.

Table 5.1. Tools for Design Refinement Phase

Tool Used	Type	Purpose
Teamwork	CASE	Algorithm description
ADAS/EDIGRAF	Attribute model creation	Parameterized, hierarchical model creation
ADAS/ADLEVAL	Parameterized model definition	Algorithm description, function library
ADAS/ADLEVAL	Static analysis	Instantiated models
ADAS/GIPSIM	Performance modeling via data flow and resource allocation	Performance estimates
ADAS/CSIM	Functional modeling and simulation	Examine functionality of design
ADAS/CSIM	Stochastic modeling	Expand classes of systems that can be modeled
ADAS/CSIM, Ada	Behavioral modeling and simulation	Compare functionality, integrate effects of subsystems, represent fault effects
Instrumented Code, Prototype Testbed	Measurement	Provide performance data for refining performance models
ASSIST/SURE	Parameterized attribute model creation/reliability modeling	Probability of system failure with respect to mission time

5.2. Design Refinement Case Studies

During Phase I, the performance and reliability modeling was concentrated on the baseline and initial design modeling phases. In Phase II, the bulk of the modeling was concentrated on initial design reliability modeling and fault tolerance modeling for the design refinement phase. As a result, an integrated fault tolerance/performance model of a prototype FTPP design was constructed and reliability analysis incorporating the results from that model was undertaken. Also, a case study of modeling operating systems was undertaken as a result both of the conclusion from Phase I that operating system effects had to be included in performance models of application algorithms and of the need to integrate operating system models into the fault tolerance/performance model.

5.2.1. Operating System Performance Modeling for Distributed Real-Time Systems

5.2.1.1. Introduction

In designing and developing a highly reliable and fault-tolerant system, the use of performance modeling at various stages of the process may provide useful information to system developers. This task investigates issues on the role of performance modeling in an integrated design environment. In particular, it examines modeling abstractions and modeling fidelity issues for incorporating operating system (OS) characteristics into system performance models. Given the application of interest, the work concentrates on real-time, distributed operating systems rather than general-purpose systems.

Specific goals and objectives were to identify useful performance abstractions for operating systems that maintain modeling fidelity and to demonstrate their use in a simple application model.

5.2.1.2. Performance Modeling

In keeping with the Working Group methodology, this report views the design process as being multiply and recursively iterative. In this context, the design process proceeds from the abstract to the specific. Performance modeling at each stage and substage can help guide the decomposition process further.

Performance can mean different things at different levels of abstraction. Higher levels of abstraction provide relatively crude measures, while lower levels can provide more accurate measures, as shown by the modeling experiments described below.

The most important question for any modeling effort is clear: how can lower level details be abstracted without losing too much fidelity in the model? In other words, what can be abstracted and what cannot? To some extent, the answer is dependent upon the importance of the issues under experimental study, i.e., the dependent variables for the modeling effort, relative to the particular system.

Since performance is only one of the many design requirements a system must meet, modeling trade-offs need to be made. One such trade-off is modeling fidelity versus cost. High-fidelity models may be expected to give more accurate results, but may be more costly to develop. Ideally, system modeling should allow developers to select a level of detail appropriate to their particular needs and constraints.

In the realm of performance modeling, the partitioning of the system is a very important issue. Different perspectives on performance issues are required to model the application, the OS system services, the OS task controls, and the architecture and its interconnection network. The performance attributes at each level of modeling abstraction vary according to the system element being modeled.

Suppose that a model of operating-system abstractions for task control is to be created. The operating system would decide when tasks get executed, blocked, or suspended. Performance metrics, such as processor utilization and task response times (which depend on OS scheduling policy), can be used. Some task control mechanisms are relatively simple and can be modeled using performance simulation tools like the Architecture Design and Assessment System (ADAS), such as fixed schedules and preemptive priority. It is also true that task control can be very complex to model, such as when modeling the Ada tasking semantics, since so much is hidden by the language and its run-time system.

If application software is being modeled, the most abstract view might use an estimate of service time, examine data flow in and out, and consider memory requirements. An intermediate level of abstraction might consider calling sequences and pseudocode for specific pieces of the algorithm being modeled. The most granular and least abstracted view might model detailed timing constraints and instruction counts. Figure 5.3 illustrates this point.

If operating system services are being modeled, the most abstract view might estimate the percentage of overhead for a task service. An intermediate-level model might evaluate performance in terms of the number of OS calls multiplied by the total amount of time per call by a task. A further degree of refinement in detail might model shared memory contention. Table 5.2 illustrates this point.

If interprocessor communication is being modeled, the most abstract view might model communication in terms of a fixed size delay. An intermediate level might take the model further and determine the delay based on channel bandwidth, size of data message, and overhead related to a network packet. Even more detail could be achieved by modeling the changes in delay dynamically. The least abstracted view would have detailed timings based on all of the factors that enter the transmission process. Figure 5.4 illustrates this point.

In summary, useful performance abstractions exist for applications, operating systems, and architecture modeling. In the case of operating-system control and interprocess communication (the subjects of the experiments described below), some control and communication semantics can be modeled relatively easily. More complex task-control semantics, such as with Ada tasking, are more difficult to model.

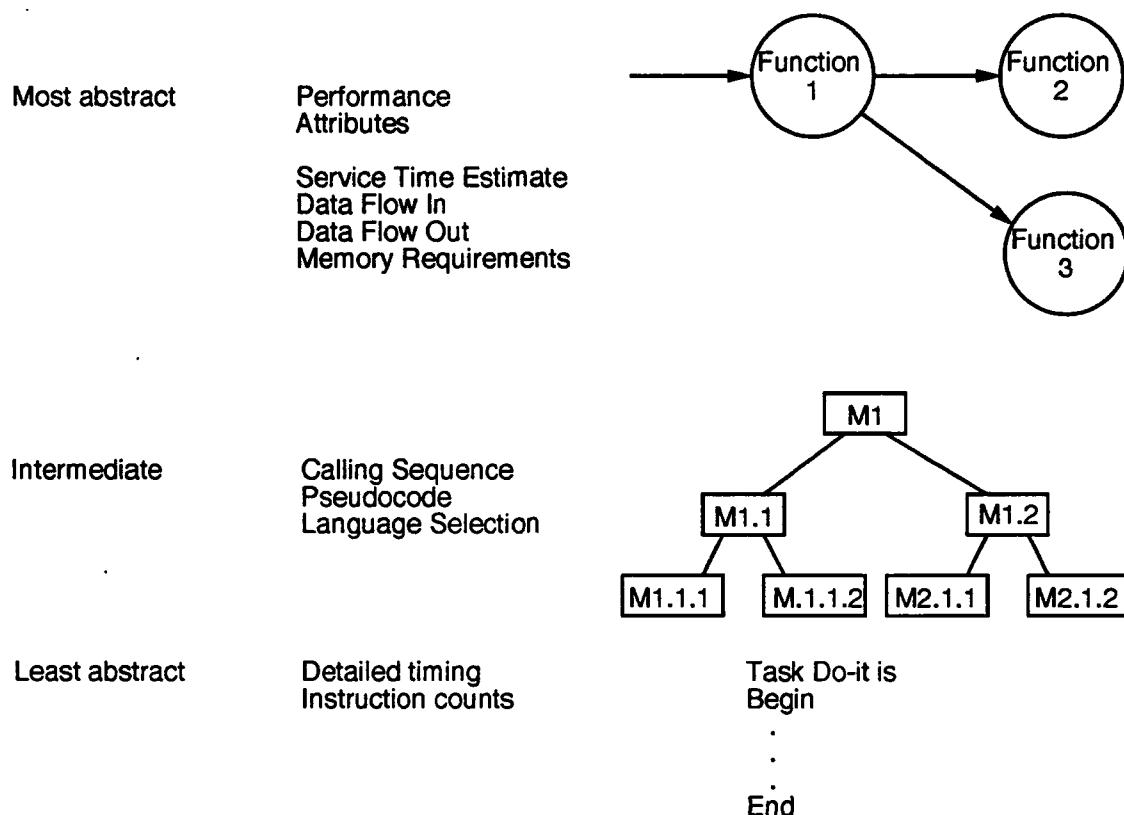


Figure 5.3. Application Software

Table 5.2. Operating Systems Abstractions for System Services

Level of Abstraction	Performance Attribute
Most abstract	% Overhead for task
Intermediate level	# OS calls × time per call by a task
Lower level	Dynamic model of OS services (e.g., shared memory contention)
Least abstract	Actual run time environment

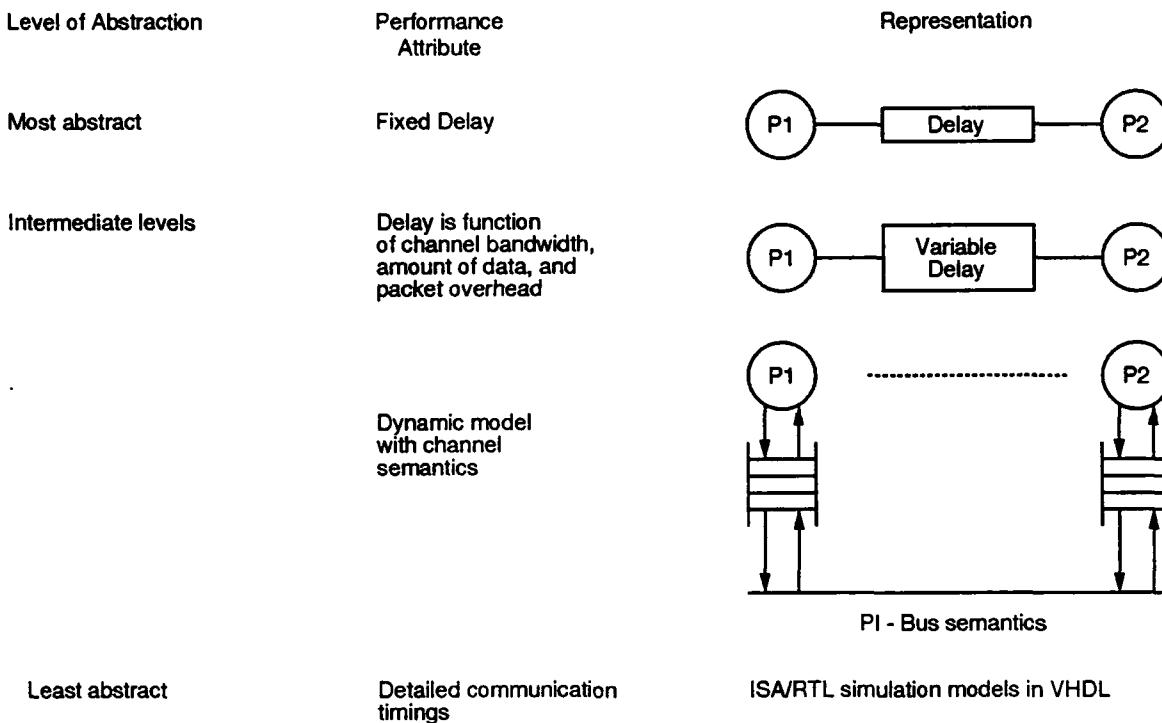


Figure 5.4. Interprocessor Communication

5.2.1.3. Experiment

This experiment illustrates the relationship of model fidelity to levels of abstraction. It shows that the addition of details to the model produces simulation results that have increasing degrees of fidelity. The experiment was constructed so that performance measures taken from models containing operating system and communication abstractions could be compared to performance measures taken from an actual prototypical implementation. The models were constructed and simulated using ADAS. They consisted of one very abstract model and a second more detailed model, which allowed for differing degrees of abstraction in functional simulation with the same topographical model. Figure 5.5 illustrates the experimental paradigm.

5.2.1.3.1. Description

The system application chosen for this experiment was a distributed client-server model. A single client module was included, with six server modules to provide generic services and a single Ethernet-like network for communication among processes. Each functional component was conceived to be executing on separate hardware processors. In simulation, the functional modules were utilizing their own processing ele-

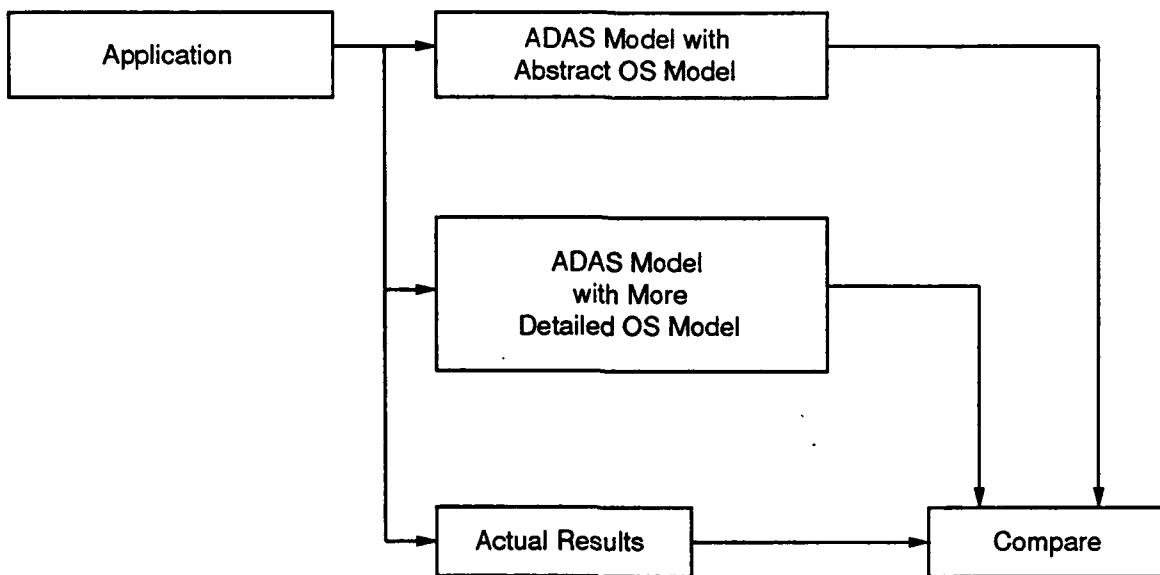


Figure 5.5. Demonstration

ments (PEs), while the network modules were viewed as competing for the Ethernet transmission medium.

As mentioned above, the ADAS models included two basic levels of abstraction, a high level of abstraction and a more detailed level, with this second level being simulated in two modes of increasing detail. The models included operating-system abstractions and communication abstractions.

With the ADAS performance and functional modeling tool, node firing delays (or the amount of simulated execution time a process uses) can be random numbers that are computed by user-defined routines during functional simulation. Random numbers are often required in simulations to model unpredictable events, such as arrival rates of requests for service and random service times.

The use of stochastic attributes to represent random events in the modeling process enhances fidelity. Random numbers are usually generated from distributions that attempt to model real-world behavior, such as Poisson distributions for queuing models, exponential and Weibull distributions for failure models, and uniform distributions for service-time modeling. A particular distribution and its parameters are chosen for the application based on experimental observations and assumptions about the processes involved in the experiment. Simulations should then be carried out until performance measures stabilize. It is difficult to predict a priori how long a simulation should run; usually the experimenter should decide based on modeling experience and knowledge of the particular modeling tool.

This experiment examined average network throughput, client utilization of its processor during simulation run, and server utilizations of their separate processors during simulation run. In this context, throughput was considered to be the total amount of data that passed out of the network nodes in a given second of simulation time. Client and server utilizations were considered to be measures of the amount of simulation time, as a percentage of the total simulation time, that the individual hardware PEs were busy.

5.2.1.3.2. Independent Variables

This experiment used three independent variables. They were the client execution time, the server execution time, and the size of the user message. The client execution time was based on measured values from an actual prototype implementation of the application. Although it turned out to be a very small amount of time, 500 microseconds, a greater time would not have affected the essential problem being modeled. It was statically set, and remained the same for all simulation runs.

The server execution times were chosen to be uniformly distributed about a mean value. The mean value was statically set prior to each simulation run, with the uniform distribution being calculated dynamically for each server node firing. The mean times varied from 500 microseconds to 1 second.

User messages were varied from 64 bytes to 32 kilobytes, with the specific size statically set prior to each set of simulation runs (ten settings). Message size was incorporated directly into the network simulation models as network node firing delays in the first model, and as production and consumption of simulation tokens in the second model.

Sets of simulation runs were carried out, with each of the ten user message sizes being held constant for all of the twelve server mean values, resulting in 120 data points.

5.2.1.3.3. Model One

Model One was the most abstract and coarsely grained of the experimental models. This model assumed that the execution times for both the client and server models encompassed the application functions and system networking functions, with additional network functions modeled as part of the network nodes. Again, firing delays for the node execution times were taken from an actual prototype application implementation.

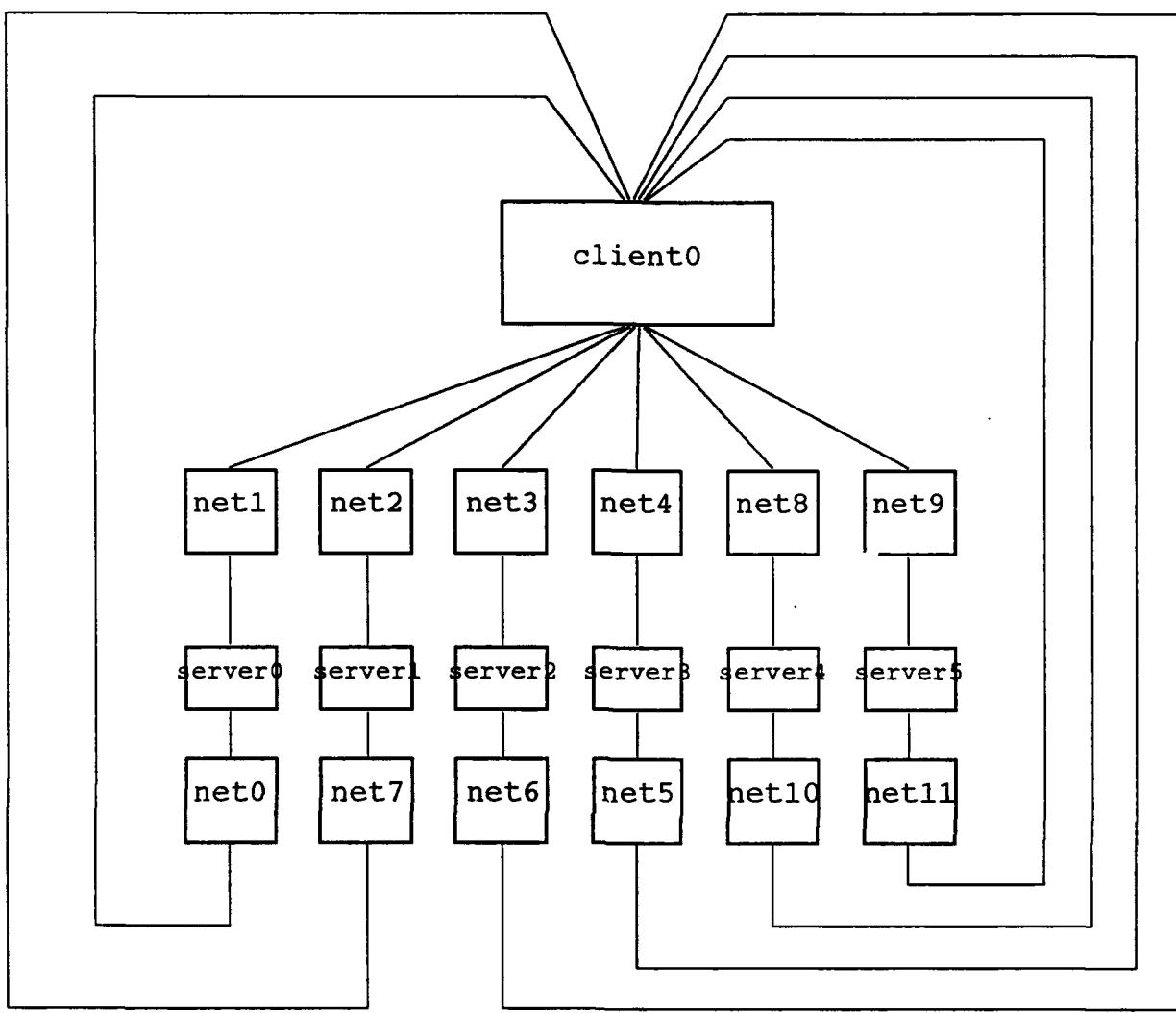


Figure 5.6. Model One

This model assumed that transmission time across the network was a function of network bandwidth and the length of a user message. The network node firing delay was set to be a function of the size of the user message as a result of this assumption. The total user message was assumed to be transmitted and received without being broken down into network transmission packets. Coarse-grained contention for network services was modeled in terms of the network nodes competing for the network hardware resource. Figure 5.6 illustrates the ADAS software graph of Model One.

5.2.1.3.4. Model Two

Model Two represented a less abstract model with the introduction of two finer levels of detail. In these two increasingly detailed views of the system, the client and server modules were decomposed into application functions and I/O functions. This more closely represents the way that the operating system and its network protocol software actually function in relationship to any specific application. The ADAS graph topography was hierarchical in its decomposition of the model, and it was the same for both versions of Model Two. Figures 5.7, 5.8, 5.9, and 5.10 illustrate the ADAS software graph hierarchy for the Model Two topography.

5.2.1.3.4.1. Model Two — First Version

The first version of Model Two made an assumption about user message transmission similar to that made for Model One. In both cases the total user message was transmitted and received as a unit. However, in this case, the transmission time was a function of the number of Ethernet packets required for a particular message size (packet sizes ranging from 64 bytes to 1024 bytes, even though Ethernet packets can range up to 1518 bytes). Based on the prototype implementation, the amount of Ethernet transmission time was calculated (exclusive of operating system network services) for messages of different sizes. The firing delay for the network nodes was statically calculated in terms of this transmission time, and the simulations were run using this calculation.

This version represented a more detailed and accurate view of data transmission in the system. Message transmission interleaving was again modeled in terms of contention for the Ethernet hardware resource. The next version refined this model further.

5.2.1.3.4.2. Model Two — Second Version

Using the same graph hierarchy as the first version of Model Two, the second version altered the network model assumptions. In this case, the user messages were decomposed into Ethernet packets for transmission, with each packet no larger than 1024 bytes (arbitrarily chosen for modeling convenience). A single token was used to model a single packet of user message data. (*Note:* Recall that in Model One and the first version of Model Two the entire user message was sent and received as a single token.) The total user message thus consisted of the number of token packets required to send the message by the actual implementation. Consumption and production of tokens from and to network nodes were scaled to the size of the

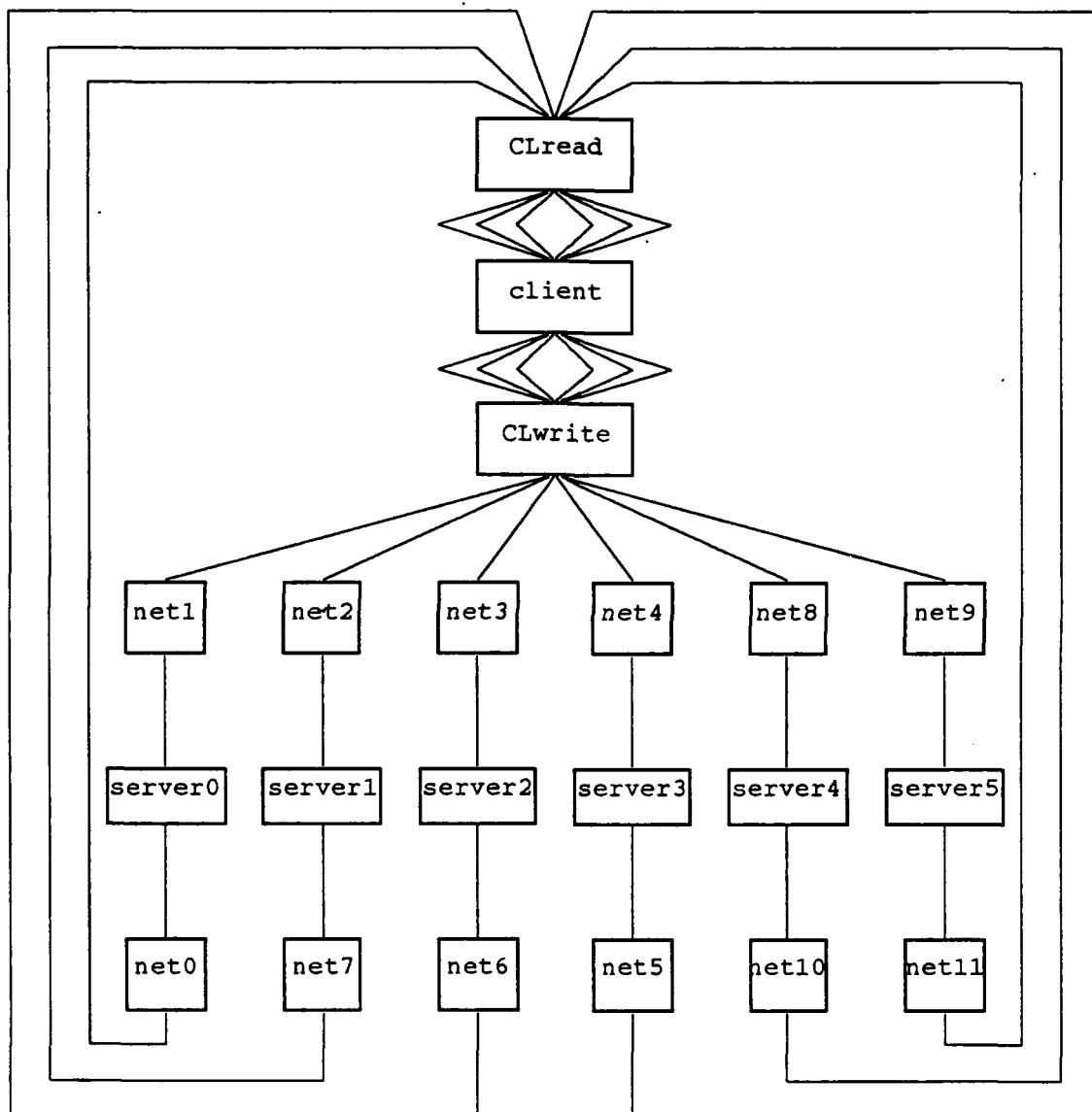


Figure 5.7. Model Two

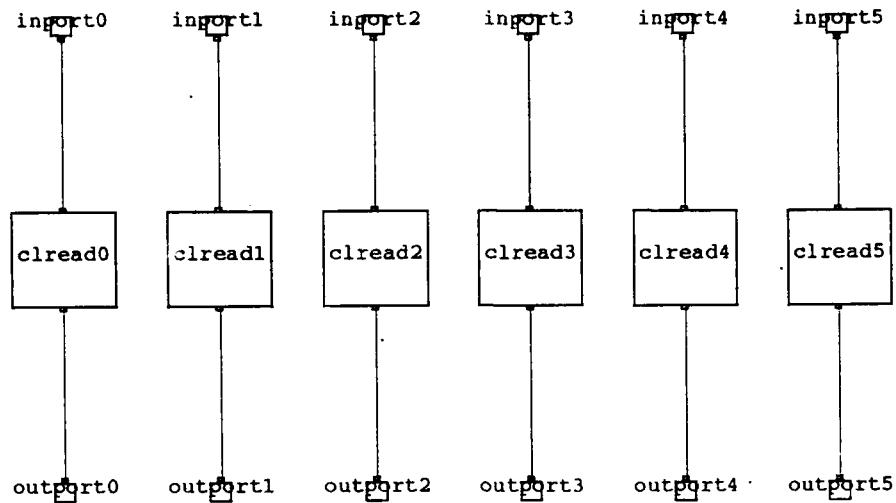


Figure 5.8. CLREAD.SWG

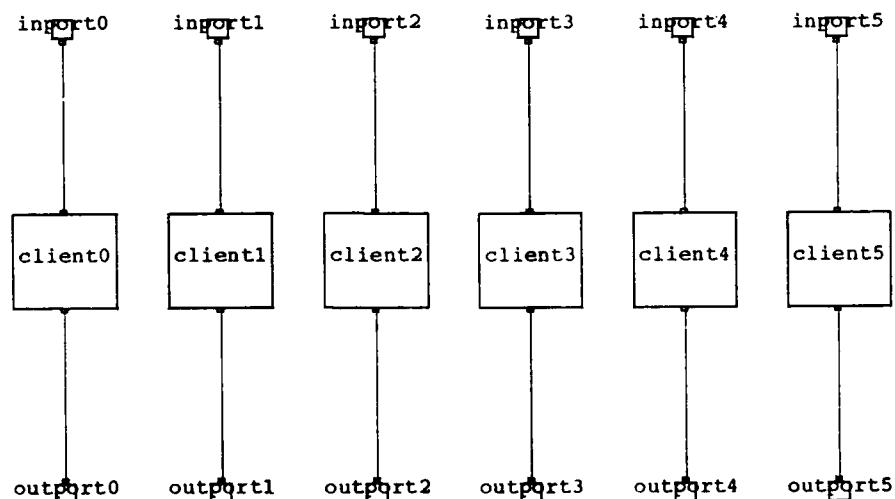


Figure 5.9. CLIENT.SWG

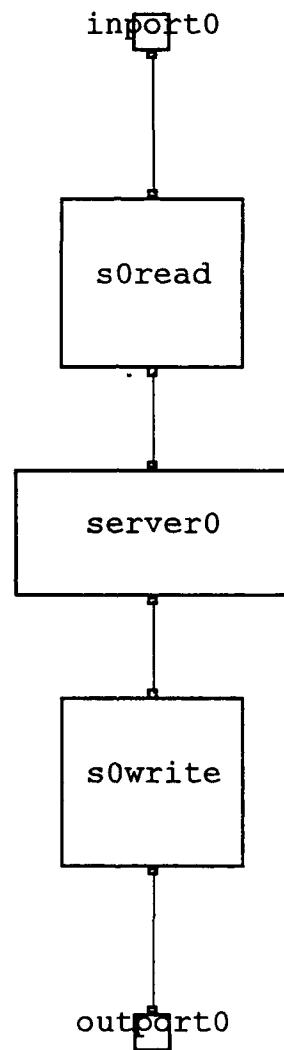


Figure 5.10. Server 0

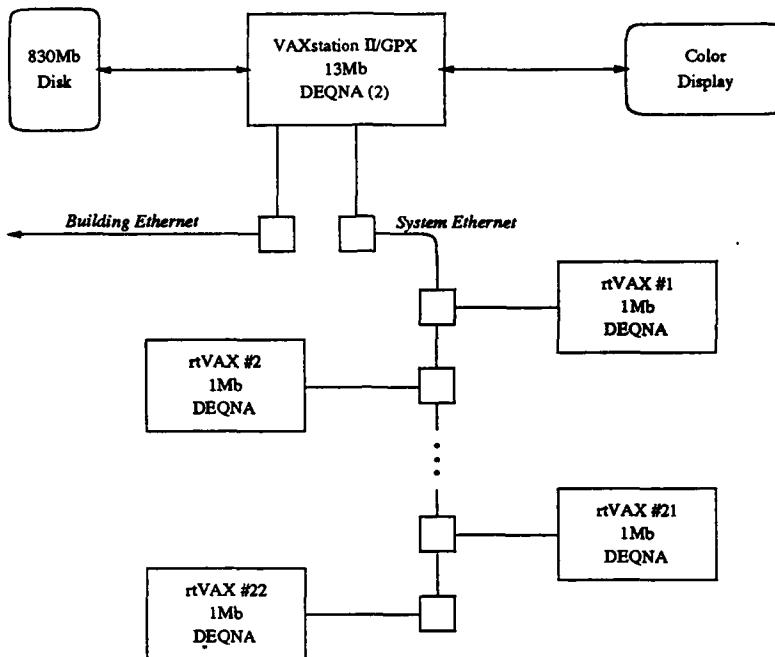


Figure 5.11. Hardware Configuration

user message (which was statically set for the entire simulation run). The client-read nodes and the server-read nodes consumed the number of tokens representing a full user message. The client-write node and the server-write nodes produced the number of tokens representing a full user message. The network nodes could only consume and produce a single token. Contention for the network hardware under these conditions simulated the interleaving of packets and the consequences of message delays in transmission for the application software in an actual application. Each network node execution time (firing delay) was based on a single packet's transmission costs.

With the modification of the produces and consumes relative to message transmission and the single token as single packet modeling assumption, this version of Model Two further refines the model of the application-OS-network interactions.

5.2.1.3.5. Prototype Testbed

The prototype testbed consisted of a VAXstation II/GPX with Ethernet interface and twenty-two rtVAX 1000 systems, also with Ethernet interfaces. As in the ADAS models, the software client and servers each ran on a separate rtVAX 1000. The testbed used the VAXELN real-time operating environment. Figure 5.11 illustrates the hardware testbed.

The software design for prototype application was similar to the ADAS graph models in concept. Task PEs, or servers, received input messages and performed a synthetic computational task with executive times set as in the ADAS models. Total time for each task was accumulated with high precision by the server process, and finally an output message was transmitted.

A LoopDriver PE, or client, caused Task PEs (servers) to repeat their operations for many iterations. The start-to-finish time for Task PEs was measured based on recording the start-time before the first message was transmitted to each PE, and recording an end-time after receipt of last return message. The difference between the start-to-finish time for each task and the total synthetic compute time was considered to be “everything else,” including I/O driver time, actual Ethernet transmit time, OS kernel time, and idle times.

A LANalyzer was used to monitor the Ethernet network, measuring the total number of packets transmitted, the number of bytes sent, and the average utilization of the network. Capture and time-stamping of the packet data allowed packet transmit times to be measured with high precision. As in the ADAS models, message size was fixed for each simulation run. Figure 5.12 illustrates the software paradigm for the testbed application. In this experiment, only one LoopDriver and six Tasks were used, but expansion is possible with this testbed.

5.2.1.3.6. Experiment Results

This experiment was intended to demonstrate that increasing refinement and detail in a model results in higher fidelity models as shown by the performance measures. The results are supportive of this assertion. While ten user message sizes from 64 bytes to 32 kilobytes were used in the experiment, three representative samples of the results are presented below, representing a small user message size (64 bytes), medium user message size (2 kilobytes), and large user message size (16 kilobytes).

The dependent variables studied in this experiment were average throughput, server utilization, relative error of server utilization, network utilization, and the relative error of the network utilization.

Throughput was taken to be the total number of accesses to the network nodes during a simulation run, scaled to kilobytes per second.

Server and network utilizations were calculated to be the percentage of simulation time that the hardware resources for each were active. Relative error for each was calculated to be the difference between the ADAS utilization values and the prototype testbed values.

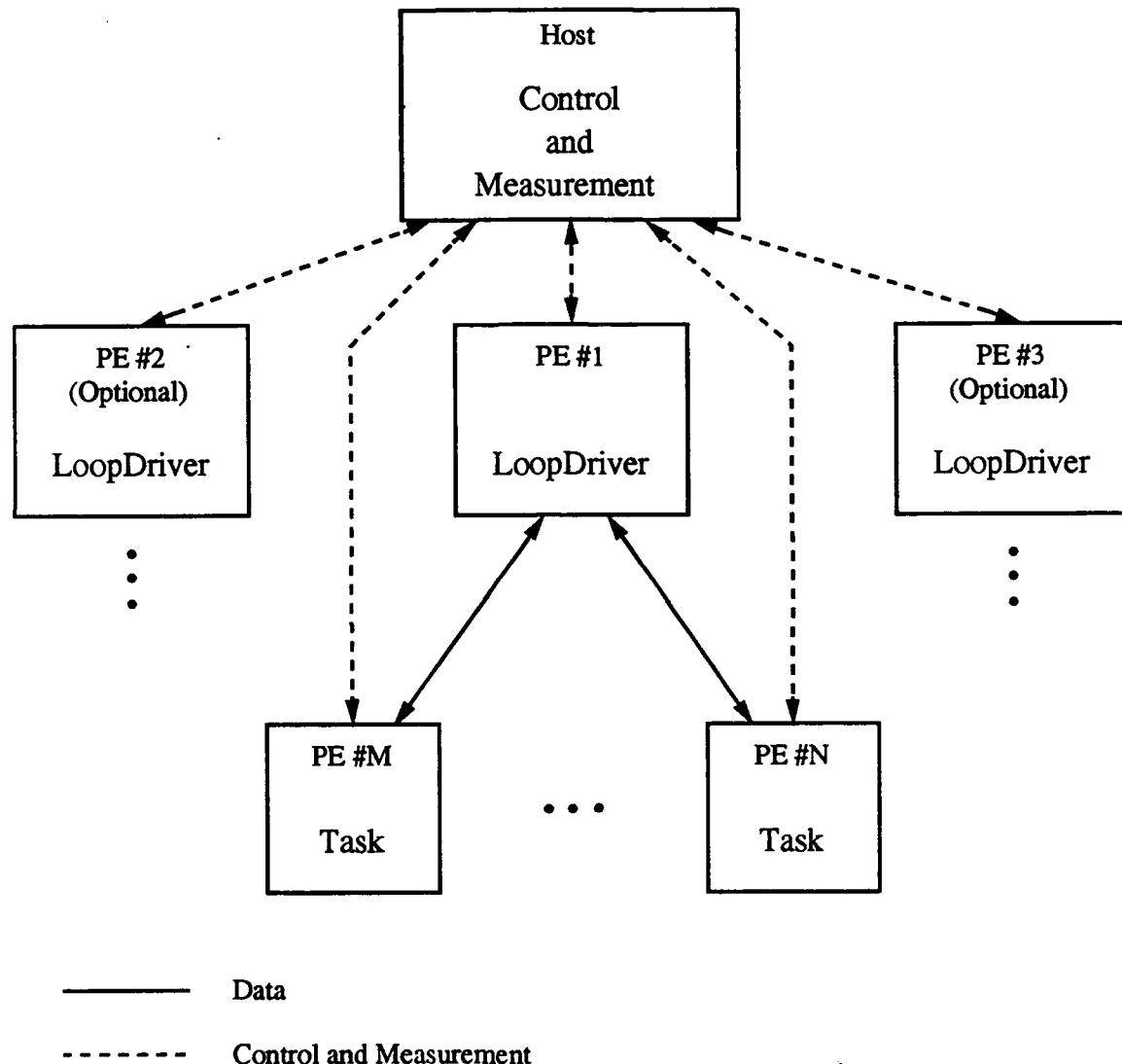


Figure 5.12. Software Design

Since sets of simulation runs were made for each user message size, each of the dependent values was plotted against the mean server time independent variable.

In the plot figures that follow, curve A represents Model One, curve B represents the first version of Model Two, curve C represents the second version of Model Two, and curve D represents the actual implementation, where

- Model One was the simplest, lumping the application and operating system together
- the first version of Model Two refined Model One by separating the I/O functions from the application
- the second version of Model Two further refined the modeling to consider the effects of Ethernet packetizing of application data

5.2.1.3.6.1. Throughput Results

Figures 5.13 through 5.15 show the throughput results for the three data message sizes. In all cases the basic shapes of the curves show that as modeling fidelity increases, more accurate simulation results are obtained. Curve C is closer to the actual implementation than the other models. Over the range of the mean server time the curves are quite different while the network transmission time dominates the server time. As the server time increases, it begins to dominate the transmission time and the curves all converge. In the case of the 64-byte message size the model overestimates the throughput while mean server times are small, and it underestimates throughput as the server times grow. Relative error with the smaller server times is high. This suggests that the model needs to be refined further to improve its fidelity. The most likely area for this refinement to be carried out is in the network services related to developing data packets for transmission. This is likely to be true for all of the results that follow.

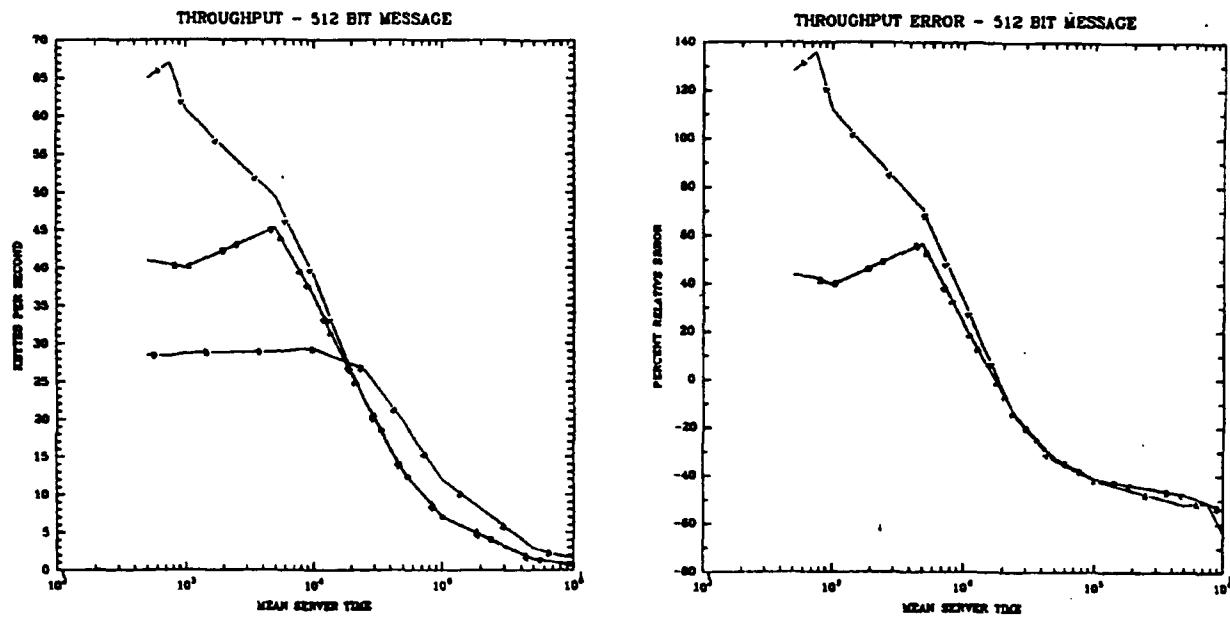


Figure 5.13. 512-Bit Message Throughput Results

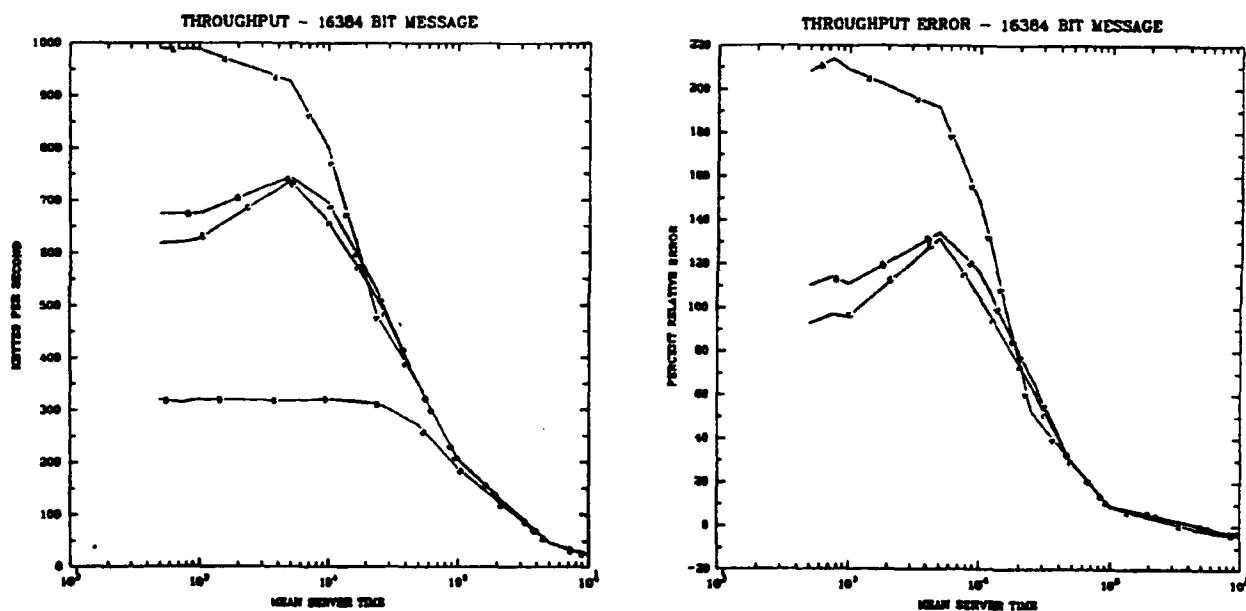


Figure 5.14. 16384-Bit Message Throughput Results

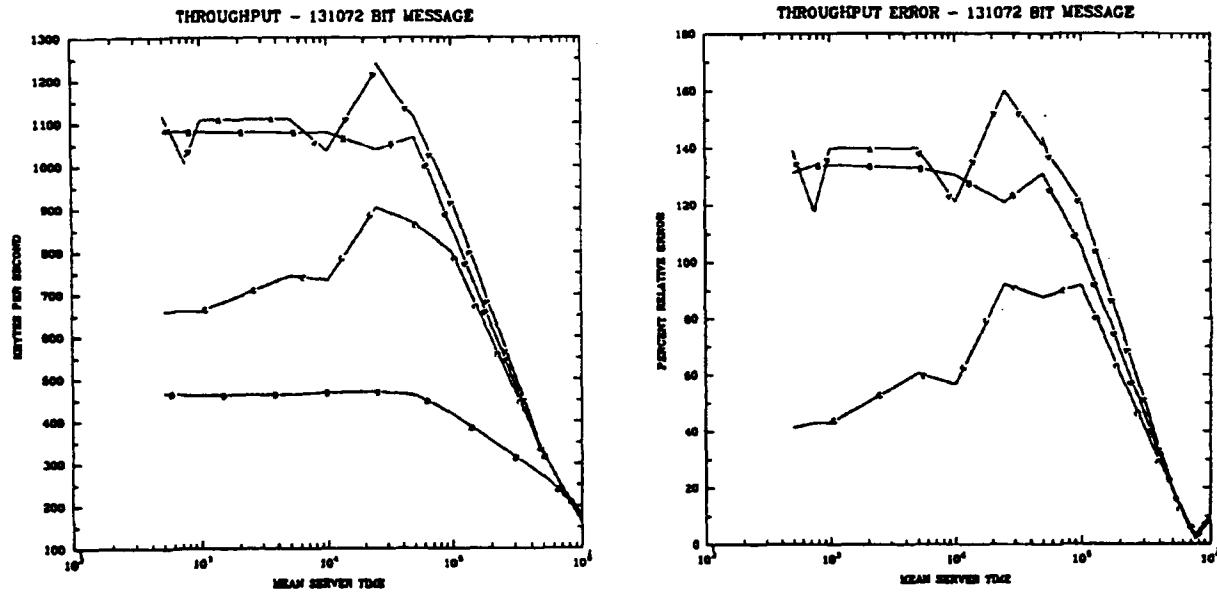


Figure 5.15. 131072-Bit Message Throughput Results

5.2.1.3.6.2. Server Utilization Results

Figures 5.16 through 5.18 show the server utilization results for the three data message sizes. As with the throughput results, the two versions of Model Two more closely approximate the results from the actual implementation in terms of the shape of the curves. However, when the relative error curves are examined, even the more refined models show a high degree of relative error. Again, the increasing detail of the models show increasing fidelity, but additional iterations to refine the model further are needed, since the models consistently overestimate the server utilization.

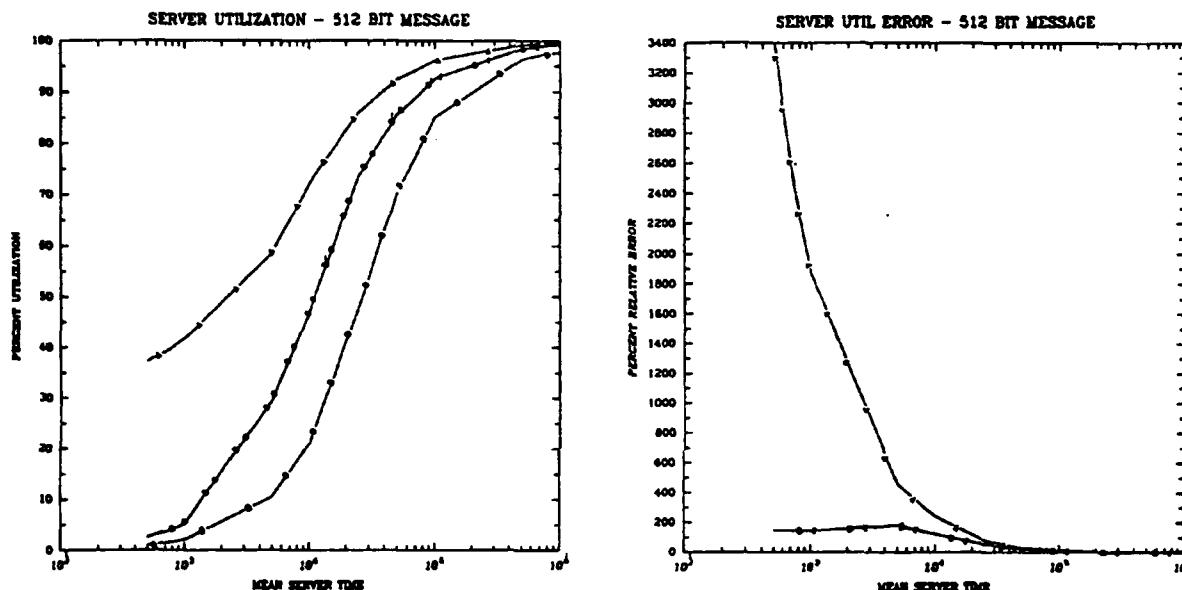


Figure 5.16. 512-Bit Message Server Utilization Results

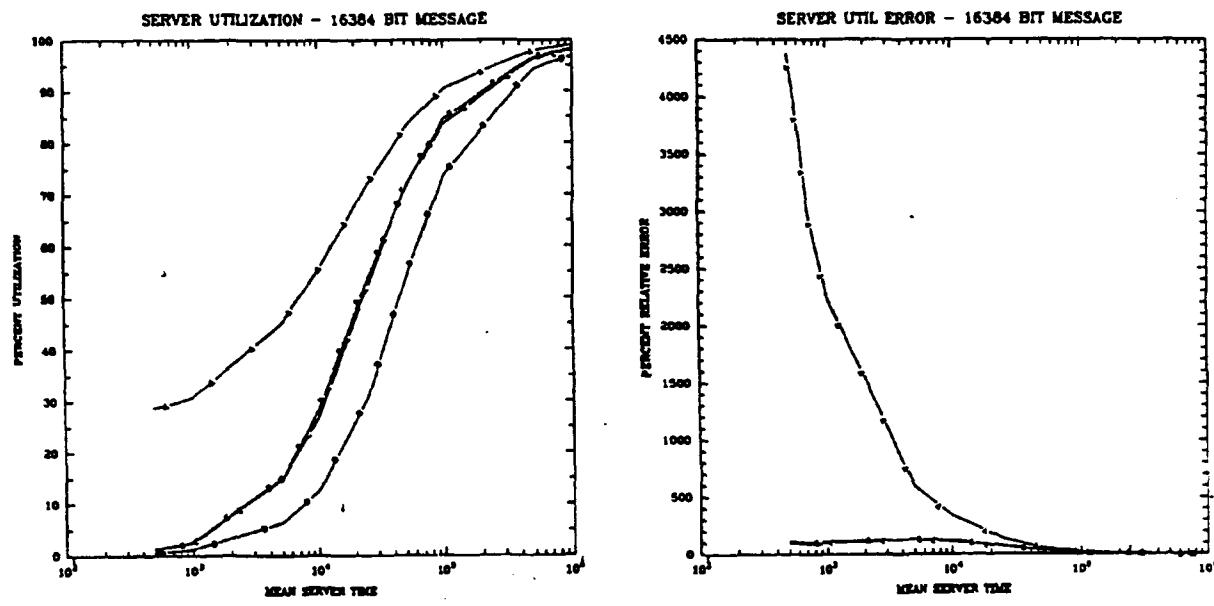


Figure 5.17. 16384-Bit Message Server Utilization Results

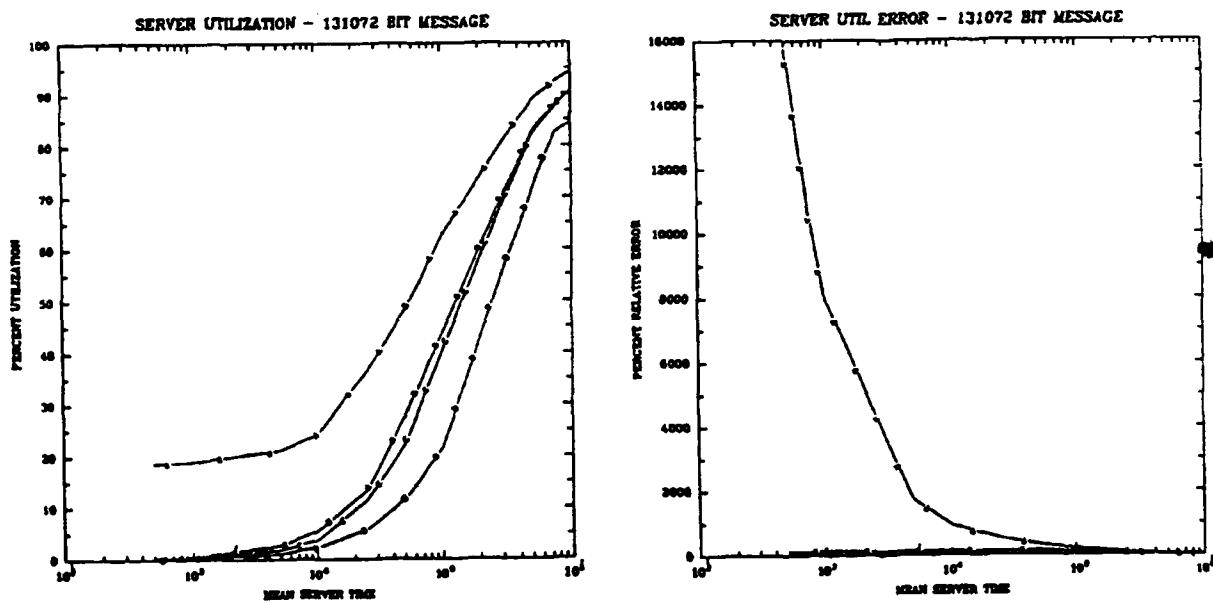


Figure 5.18. 131072-Bit Message Server Utilization Results

5.2.1.3.6.3. Network Utilization Results

Figures 5.19 through 5.21 show the network utilization results for the three data message sizes. These results are consistent with the previous sets. For small mean server times, the models overestimate the network utilization. They then underestimate it for larger server times. These results also show that the most abstract model is the least accurate. Relative error remains higher than desirable, suggesting that further refinement be carried out on the model.

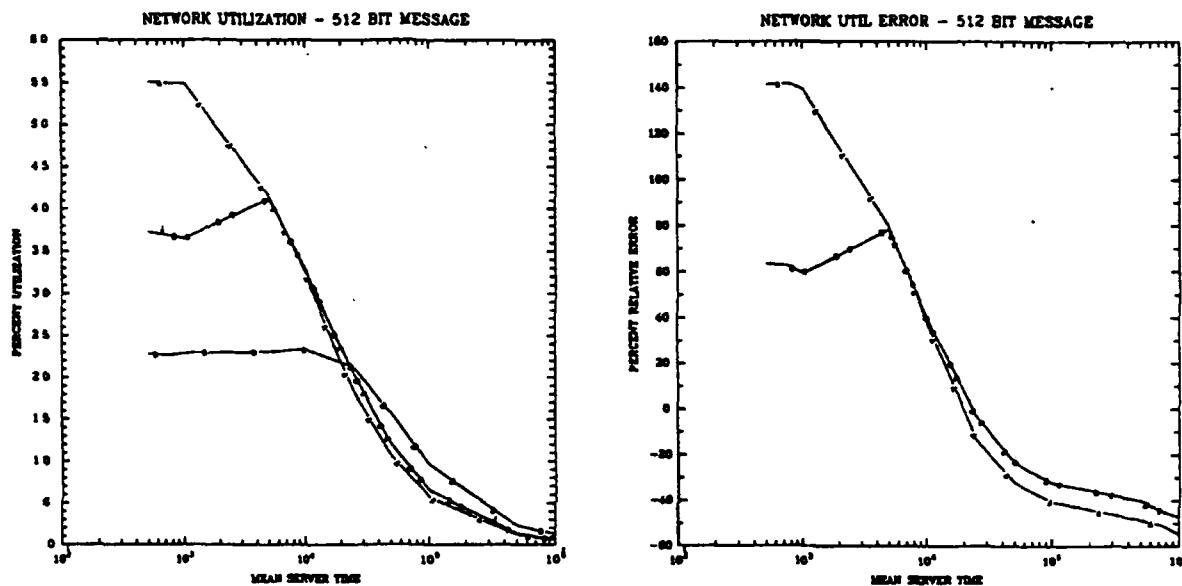


Figure 5.19. 512-Bit Message Network Utilization Results

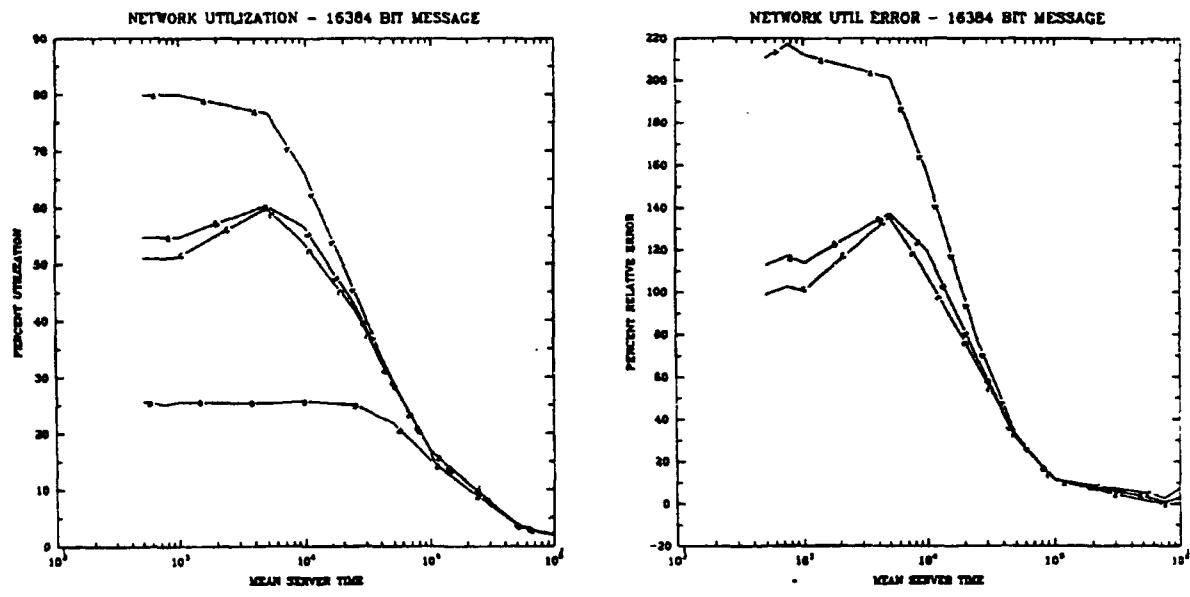


Figure 5.20. 16384-Bit Message Network Utilization Results

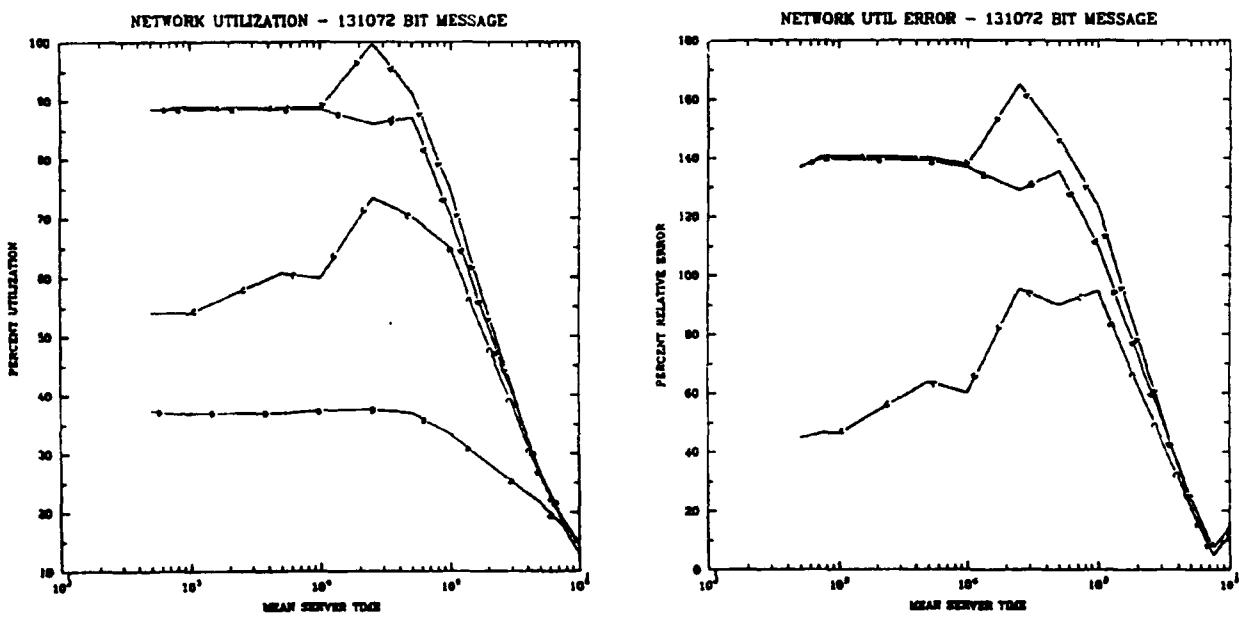


Figure 5.21. 131072-Bit Message Network Utilization Results

5.2.1.4. Experiment Conclusions

This experiment indicates that the hierarchical refinement approach is a useful way to carry out a modeling effort. It suggests that any modeling effort should strive to achieve the simplest model that will give acceptable results. Clearly there will be trade-offs in cost and effort for a level considered acceptable by the modelers in any specific case.

It is also clear that any model must be validated for it to be useful. Validation implies that the model must be compared against reality to establish a range of valid results. This helps to identify problems in the model or in the measurements that were used with the model. Accurate timing information is essential to a good simulation model and this often requires specialized equipment. A model is only as good as the information it uses to generate results.

Modeling efforts are most useful when the simulation results can be reused. In other words, the effort should seek to develop a reusable model that can become a library element after validation. This lowers the cost of any modeling work over time.

Experience and judgment are important in the modeling and analysis process. Any modeling tool is only as good as the information it has available to it, and that information is a product of the professionals involved.

5.2.2. Behavioral Modeling for Fault Tolerance Evaluation

5.2.2.1. Background

In Phase I, the selected case studies had emphasized modeling the performance characteristics of highly parallel computing architectures. The primary emphasis of Phase II was reliability analysis for highly parallel computer architectures and the modeling and evaluation of fault tolerance mechanisms that are inherent in highly reliable architectures. This case study was undertaken to investigate a process by which the effectiveness of fault tolerance mechanisms can be evaluated during early- to mid-design phases and to demonstrate the use of the results from this process in the evaluation of system performance and reliability.

The design approaches that lead to extended system life and improved reliability such as sparing and redundancy are conceptually simple. However, to be effective, the mechanisms that support these concepts must also be highly reliable. Therein lies the difficulty in implementing effective fault tolerance mechanisms for complex systems and the need to emphasize the design and assessment of fault tolerance throughout the system development cycle. The functionality, effectiveness, and performance of fault detection, containment, identification, and recovery mechanisms must be systematically evaluated during development to assure that objectives can be met by candidate designs, to identify and remove design flaws, and to support design trade-offs between fault tolerance and other system characteristics.

Among the important characteristics or attributes that must be addressed in the design and evaluation of fault-tolerant systems are:

- fault assumptions for all mission phases
- fault classes and error characteristics
- fault isolation techniques
- fault containment boundaries
- error detection and fault diagnosis elements of the architecture, including those that check the fault detection and reconfiguration mechanisms
- assumptions made for the effectiveness and detection times associated with error detection mechanisms
- error recovery and system reconfiguration processes and mechanisms
- synchronization characteristics for redundant processing components

- provisions for fault-tolerant power and clocking
- Byzantine Resilience
- degree of independence of fault tolerance mechanisms from the application and operating system software

Neither performance modeling, which focuses on workload, nor reliability modeling, which focuses on transitions from operational to failed states, is sufficient to fully evaluate the functionality, effectiveness, or appropriateness of the fault-tolerant attributes of a system. Techniques including, but not limited to, inspection, design policies, and detailed design simulation must be employed as appropriate for each of the above-listed attributes. This case study was directed toward the use of models which can be used to represent and simulate the behavior of the fault masking, detection, identification, and recovery mechanism at the level of detail available in the early- to mid-design phases. In addition to workload and operational failed states, the models employed must represent the functionality of each system element under normal and faulted conditions. Besides functionality, specific data and control variable values, as well as data and control flow, become important. Since fault tolerance mechanisms typically engage the resources of all elements of a system architecture, the effects of the appropriate system elements such as the application software, the hardware architecture, the operating system software, and the specific hardware and software fault detection, masking, diagnosis, and recovery elements must be modeled. Since these elements interact, it is not sufficient to model each element separately. An integrated model that incorporates the effects of all appropriate system elements must also be utilized to fully assess fault tolerance mechanisms. This model captures the behavior of all appropriate system elements. It could be constructed from submodels of each system element which could be analyzed separately but could also be combined to simulate the interactions of the system elements represented by the submodels.

By providing the capability to represent faults at a functional level, the following items are among the characteristics of system behavior that can be studied:

- how faults in one processing node are reported to other processing nodes
- which fault types are signaled to other nodes
- which local faults require global actions
- how application tasks are affected by fault recovery
- how computing delays and lost information are compensated for in application tasks

- how data consistency is maintained
- how redundant system components are managed
- how spare system components are managed
- how functions are reassigned to non-faulty nodes
- how critical functions are protected
- how shutdown and restart mechanisms and commands are verified
- task scheduling
- self tests
- how state is aligned and processors resynchronized after error recovery procedures
- the potential domino effect in global recovery strategies
- the effectiveness of error detection, error recovery, and system reconfiguration
- times required to detect, recover, and reconfigure
- the detection of faults in the fault tolerance mechanisms

A prototype version of the Fault-Tolerant Parallel Processor (FTPP) under development at C.S. Draper Laboratory was designated as the architecture to be used in this case study. Factors contributing to selection of the FTPP architecture were

1. the FTPP was the only fault-tolerant parallel architecture for which appropriate information was available;
2. the design information available was judged consistent with early- to mid-design phases; and
3. the FTPP could support mixed redundancy levels and spare processors.

The available description of the prototype FTPP architecture [4] was sufficient to establish the functionality of all major elements and to develop models of the FTPP message transport circuits, error detection and reporting circuits, fault masking circuits and system configuration control function. The models that were developed were at or near the register transfer and logic block level. Pseudocode for the reconfiguration function and descriptions establishing the functionality for the operating

system scheduler, the network communication services, and the fault detection and identification diagnosis were provided by the C.S. Draper Laboratory.

Behavioral models were built for an application process, the FTPP architecture with fault tolerance mechanisms, the operating system scheduler, the network communication services, the fault detection and identification process, and the system recovery and reconfiguration process. These models captured the functionality, data and control flow, data and control variable values and the workload parameters for each major element. Models were tested separately to determine if they behaved according to the description for these elements. An integrated system model which combined these separate submodels was used to simulate the behavior of the system under both fault-free and faulted conditions. A number of test cases or experiments were set up and carried out using the integrated model. Analyses of the experiment results were used as part of a fault tolerance evaluation and were used to support performance and reliability analyses. In addition, performance analysis was used to provide workload portions of the behavioral models. Figure 5.22 illustrates the relationship of the behavioral modeling used in this case study to other modeling activities. Finally, the process used in this case study was examined with respect to the information required to carry out the process, the problems associated with the process, the limitations of the process, the types of models required, the types of analyses and evaluations appropriate for the process, the need for automated tools to support the process, and how this process could support a methodology for the design and evaluation of systems during the targeted design phases.

5.2.2.2. Model Descriptions

5.2.2.2.1. FTPP Overview

The FTPP architecture consists of a partially connected network of clusters of multiple primary fault-containment regions (PFCRs) consisting of an input/output element and multiple processing elements connected to a network element. The network elements are specialized hardware components that control the flow of message traffic through the system and execute the synchronization, voting, and consistent ordering protocols required to achieve Byzantine Resilience. They operate in cycles consisting of frames to recognize and classify exchange patterns, to decide which messages to transmit and, finally, to transmit, vote and receive messages. The configuration of processing elements into computational sites is constrained by the necessity of allocating each channel of a redundant processing (fault masking) group to a different network element so that each channel will belong to a separate PFCR. Multiple interconnected clusters can be built using the special input/output elements that are attached to each of the network elements in a cluster to implement the connection. In

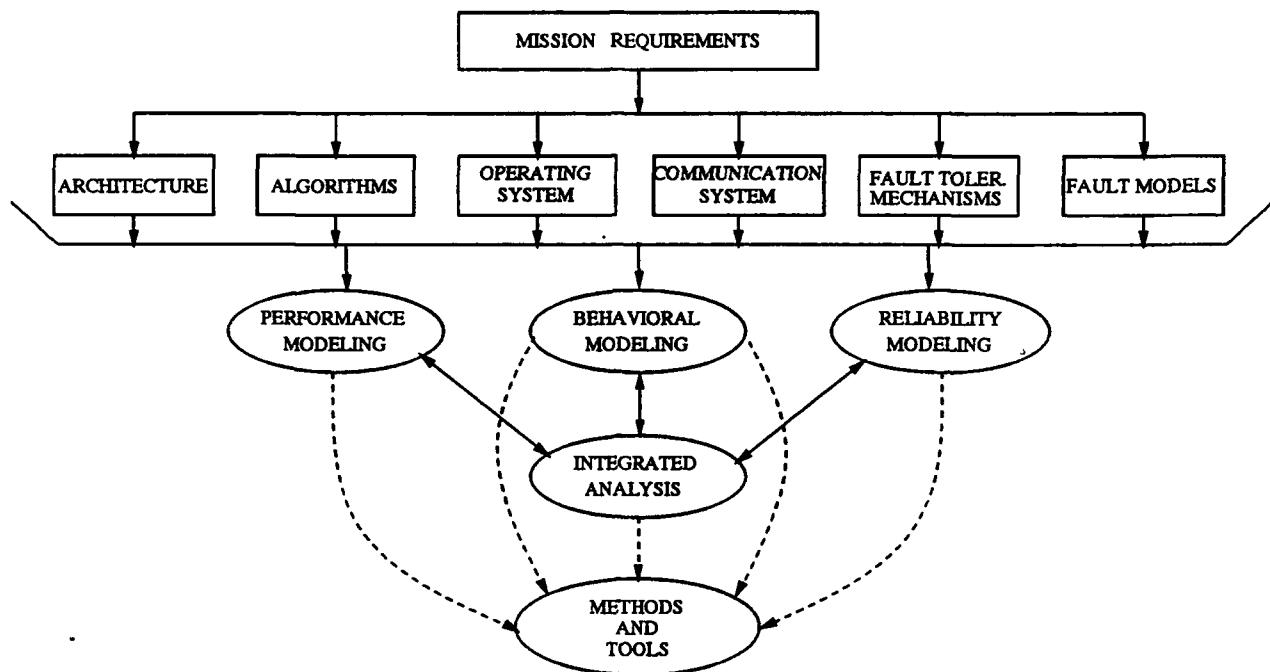


Figure 5.22. Models for Fault Tolerance Analysis

the event of a processor failure in a fault masking group, the FTPP can reconfigure to replace the faulty processor.

The FTPP is a message-based architecture, relying on the exchange of messages for communication between parallel tasks, for fault detection and isolation, and for synchronization for both fault tolerance and parallel computation. In the FTPP there are sixteen message exchange classes of two basic types. Messages that emanate from a source sufficiently redundant that a vote will guarantee receipt of identical information at all destinations are Class 1 exchanges. Those from a single source are Class 2 exchanges. A Class 1 exchange thus originates from a FMG, consists of a broadcast among all network elements associated with the source and destination FMGs and a vote on the received message, and can be completed in one round. A Class 2 message exchange is originated by a single processor and requires two rounds of voting to guarantee that all destinations receive the same data in the same order. It consists of a broadcast among NEs, a reflection of the received message, and a vote on the reflected message, and thus requires a two-round interactive consistency protocol. Therefore, the redundancy management overhead is dependent upon the mix of exchange classes utilized in the application. The different types of messages that constitute the FTPP network traffic are summarized in Table 5.3.

The FTPP configuration selected for this case study consists of one cluster containing

TYPE	FUNCTION
LERP	Exchange FIFO, Destination Class and VID
SERP	Voted Exchange of All FIFO, Destination Class and VID
Class 0	Functional Synchronization
Scoop	Functional Synchronization with Data
Class 1 within VID	Vote Partial Results
Class 1 VID to VID	Communication Between Processors
Class 2	Distribute Simplex Data

Table 5.3. Types of FTPP Network Traffic

4 PEs per NE (PFCR). This configuration is shown in Figure 5.23. In this Figure, the labels PE_i and jN indicate that processor i is assigned to channel N of virtual group j . For example, PE04 is assigned to the B channel of virtual group 0. This configuration is the same as the configuration described in Section 4.2.3 for the reliability analysis case studies. It has four fully-interconnected network elements (NEs) and sixteen processing elements (PEs).

The FTPP operating system assigns processing elements to virtual groups according to mission requirements. These groups, designated by identification numbers (VIDs), can be quadruplexes, triplexes, duplexes, or simplexes. The scenario chosen for this case study required a configuration of at least four triplex VIDs. The processors not included in these triplexes were designated simplex VIDs and were used as spares for the triplex VIDs.

The integrated fault tolerance performance model created for the FTPP consists of five submodels: an application model, an operating system model, a fault detection and isolation model, a reconfiguration model, and an architecture model.

5.2.2.2. Application Model

A simple fork and join process was chosen as the application to be modeled. This process receives data from an external source, such as a sensor, collects the data in a matrix, partitions the matrix into submatrices, distributes each submatrix for parallel computation, collects the results of the distributed computation, and reassembles that data into a new matrix. This process is illustrated in Figure 5.24. For this application, there are 6 subprocesses, labeled P1 through P6. These processes are mapped to four

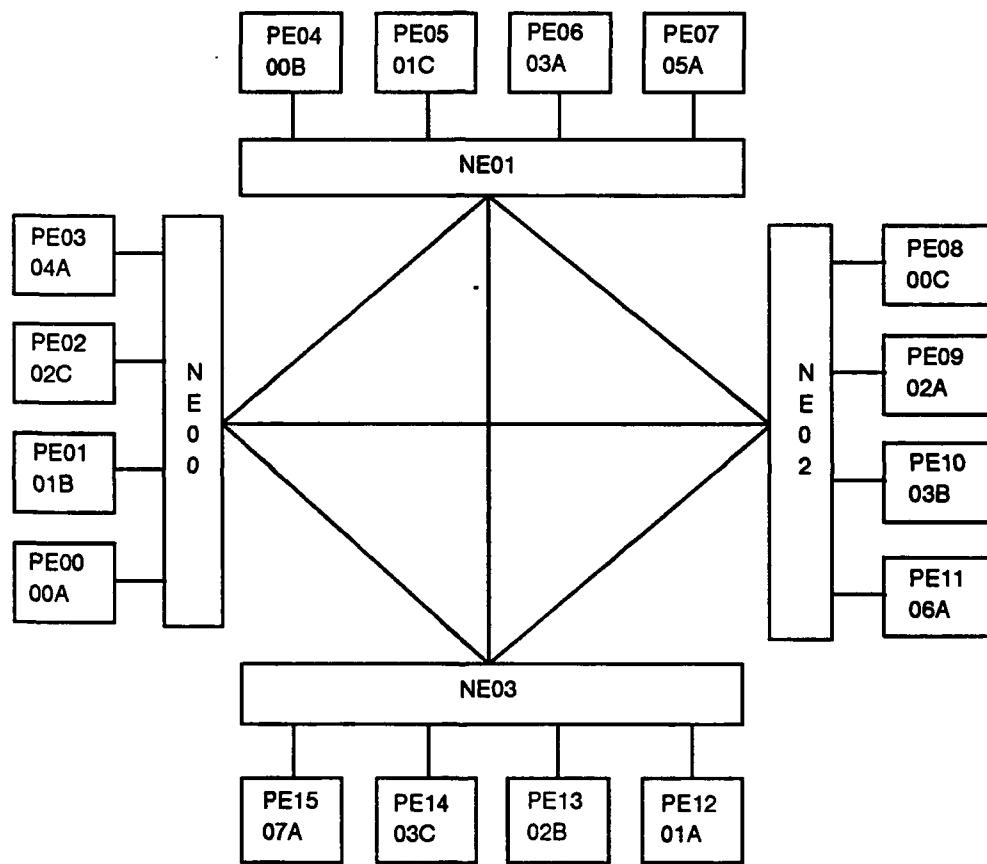


Figure 5.23. Selected FTPP Configuration

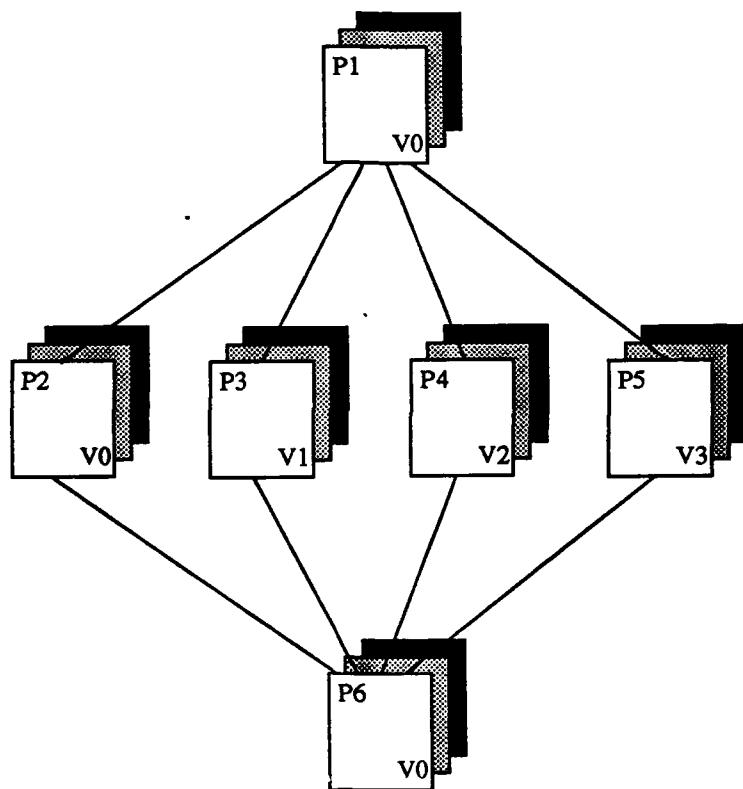


Figure 5.24. Fork and Join Application

FTPP VIDs, labeled V0 through V3. Each VID is a triply redundant processing site, and each of the three processors in a VID is attached to a unique network element. Process P1 collects external data from a sensor source, distributes it to all members of V0, partitions the data into four portions, and distributes the appropriate portion to each of the four VIDs. Processes P2 through P5 perform a computation on the distributed data and deliver the results back to VID0. P2 is assigned to V0, P3 to V1, P4 to V2, and P5 to V3. Finally, P6, executing on V0, collects the results from P2 through P5 and performs the final computation.

This application was chosen because it encompasses two types of message distribution: the distribution of data within a VID and distribution of data between VIDs. Additionally, the distribution of data within a VID contains the distribution of data from a simplex source. This allowed the modeling to cover the message class types that exist for the FTPP. The size of the input data matrix was selected to be small enough for reasonable simulation times while generating enough message traffic to trigger the fault detection and isolation processes of the FTPP. The message traffic associated with this application is listed in Table 5.4 by message type and quantity.

Table 5.4. Fork and Join Message Traffic Summary

Function	Message Type	Quantity
Distribute Sensor Data	Class 1	4
	Class 2	4
Distribute Matrices	Class 1	6
Vote Partial Computations	Class 1	8
Collect Results	Class 1	3
Functional Sync	Class 0 or Scoop	0 (Assumed in Scheduling Loop)
Totals: Class 1 = 21, Class 2 = 4, Messages = 25		

To create a model of this application, the processes were broken down into components that reflect the message distribution processes of the FTPP architecture. The control flow for these processes for each VID was then established and the model illustrated in Figure 5.25 was created.

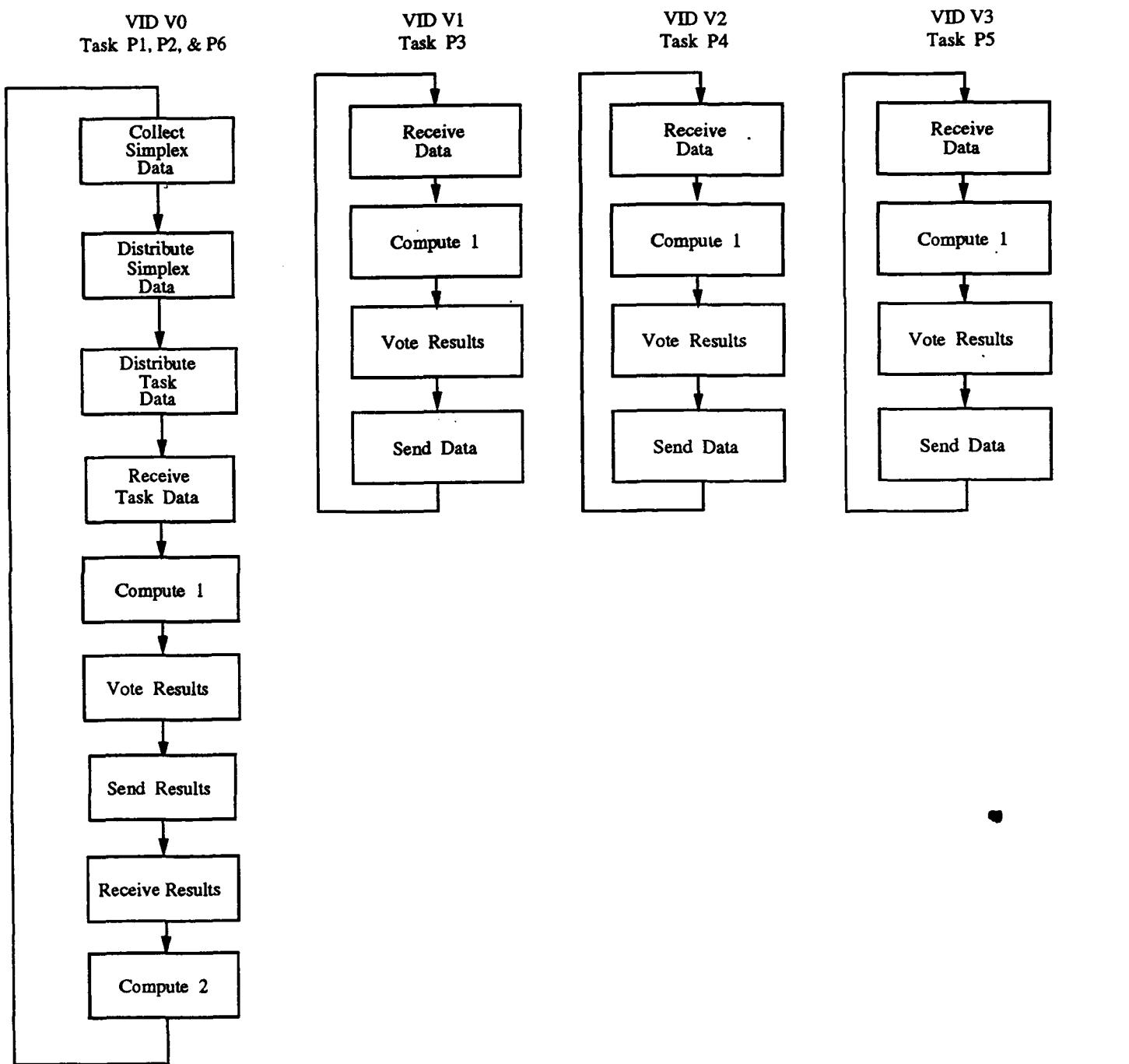


Figure 5.25. Fork and Join Application Model

5.2.2.2.3. FTPP Operating System Models

The FTPP operating system is a distributed control system. Each processing element is controlled by its resident operating system. The processing elements interact via messages to synchronize, to diagnose faulty processors, and to reconfigure the system. The operating system processes that were modeled include those that process messages for transmission to and receipt from the network core, that schedule tasks, and that synchronize members of fault masking groups.

The FTPP processors use a round-robin, non-preemptive schedule. The processing frame consists of application, timekeeper, fault detection and isolation, reconfiguration, and synchronization tasks. Each task is initiated once each scheduling loop and executes until it suspends itself and returns control to the next task in the frame. The scheduler model is illustrated in Figure 5.26.

One processing element is designated the timekeeper, but any other processing element can be assigned this function during system reconfiguration. The designated timekeeper (master) performs the following functions:

- synchronize the members of the VID and read the time value maintained by the interrupt service routine
- exchange data
- average the time and broadcast it if the resulting value is different from the previous time
- if a new time is broadcast, receive the broadcast time and update the system-wide time

The other processing elements (slaves) perform the following functions:

- receive the broadcast time
- update the system-wide time

The control flow model of the slave timekeeper function is illustrated in Figure 5.27; that of the master timekeeper in Figure 5.28.

The tasks executing in the processing elements of the FTPP communicate via messages. When a task is ready to transmit or receive a message, it invokes the network

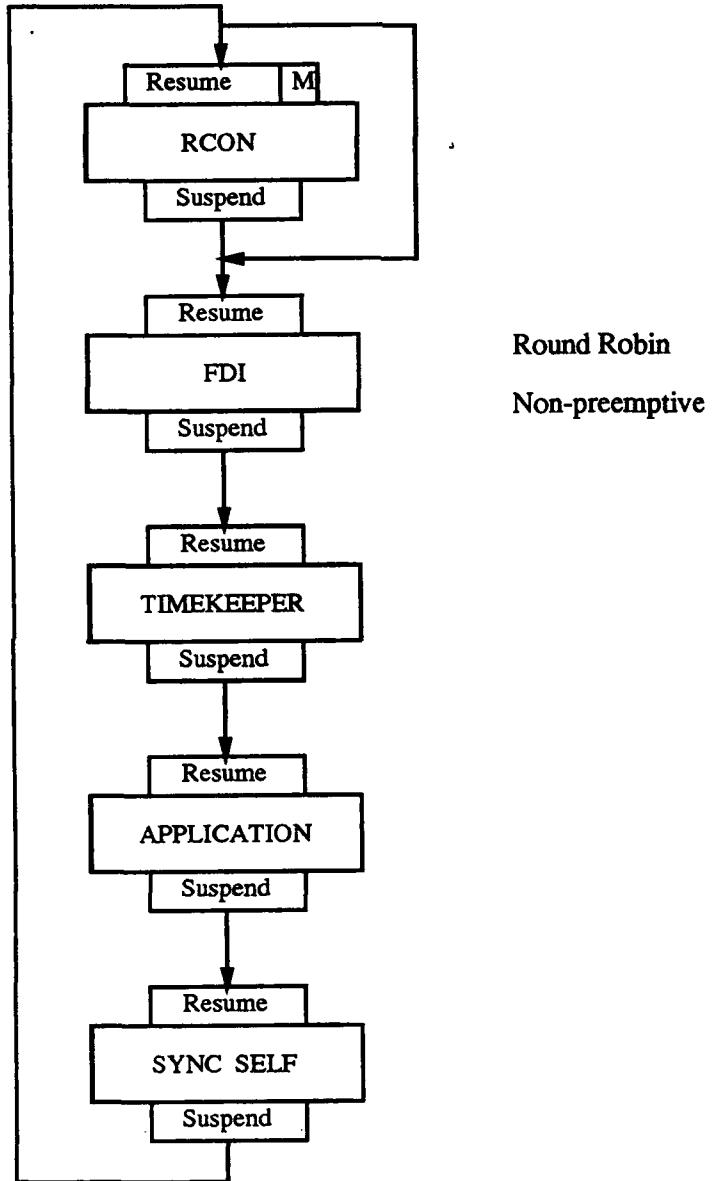


Figure 5.26. FTPP Scheduler Model

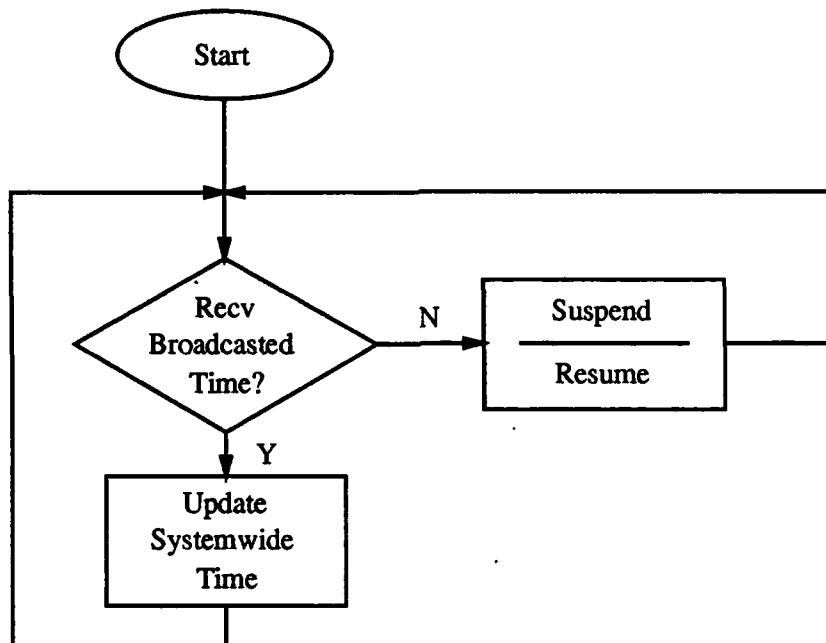


Figure 5.27. Slave Timekeeper Model

services function of the FTPP operating system. These services prepare outgoing messages for transmission, place outgoing messages in the appropriate transmit FIFO, fetch incoming messages from the appropriate receive FIFO, prepare incoming messages for delivery to the destination task, and log the syndrome information appended to the message by the voter. An outgoing message has to have header information including a message class identifier and the destination. A CRC error code is computed and appended to the message, and the message is broken into packets for transmission. An incoming message has to be routed to the designated task and reconstructed from the received packets. The top level of the network services model is illustrated in Figure 5.29. The lower-level models of the Send Message and Get Message components of the top-level model are illustrated in Figures 5.30 and 5.31, respectively.

5.2.2.2.4. Fault Detection and Identification Models

When a message is sent through the network element core, all copies of that message are voted and the results of the vote are used to create an error syndrome. This syndrome is appended to the voted message that is delivered to the indicated destination. When the message is received at the destination, the syndrome is copied into a syndrome log. When the fault detection and isolation (FDI) process is invoked, the log is checked for a minimum number of syndromes. If that minimum number of

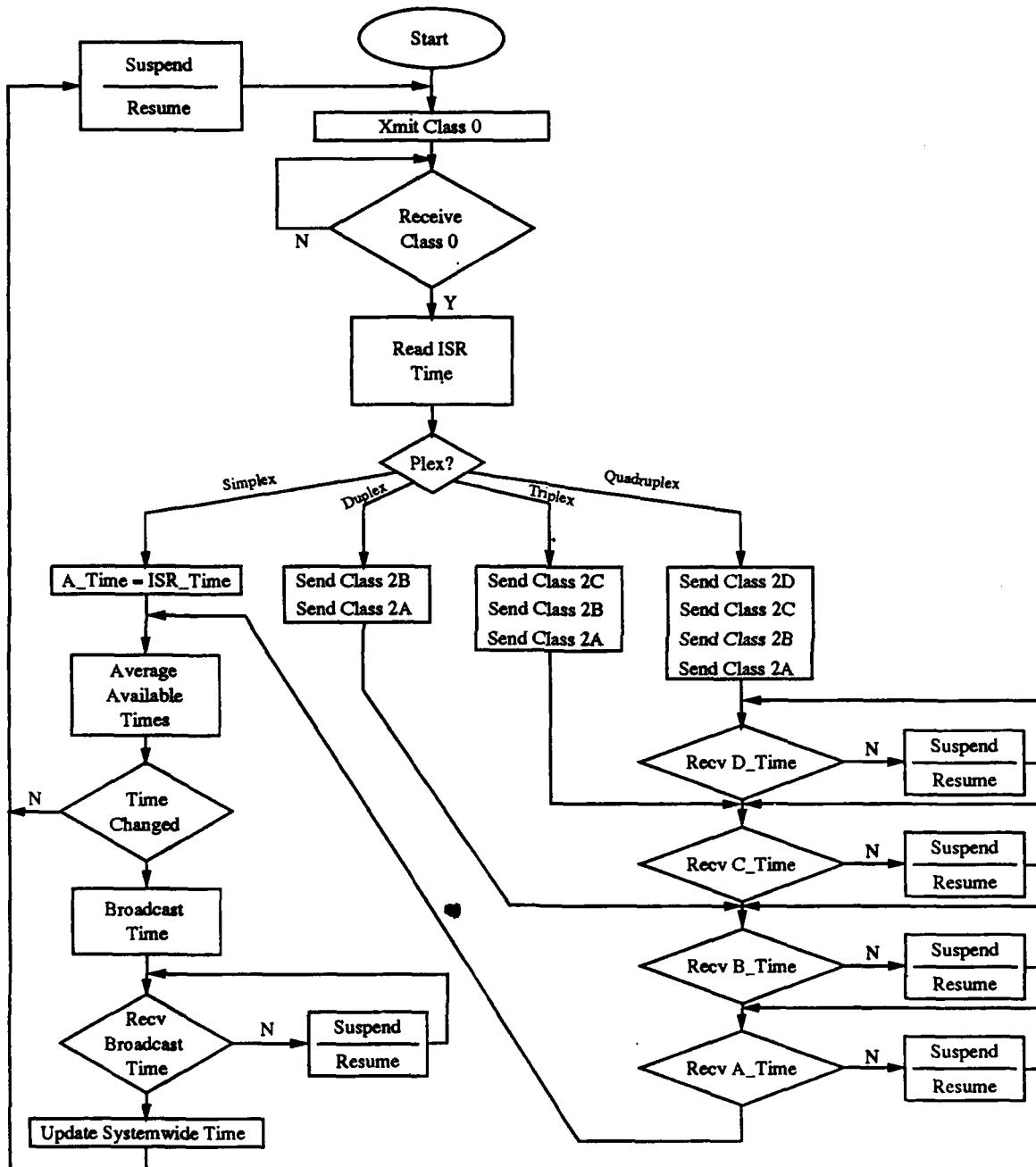


Figure 5.28. Master Timekeeper Model

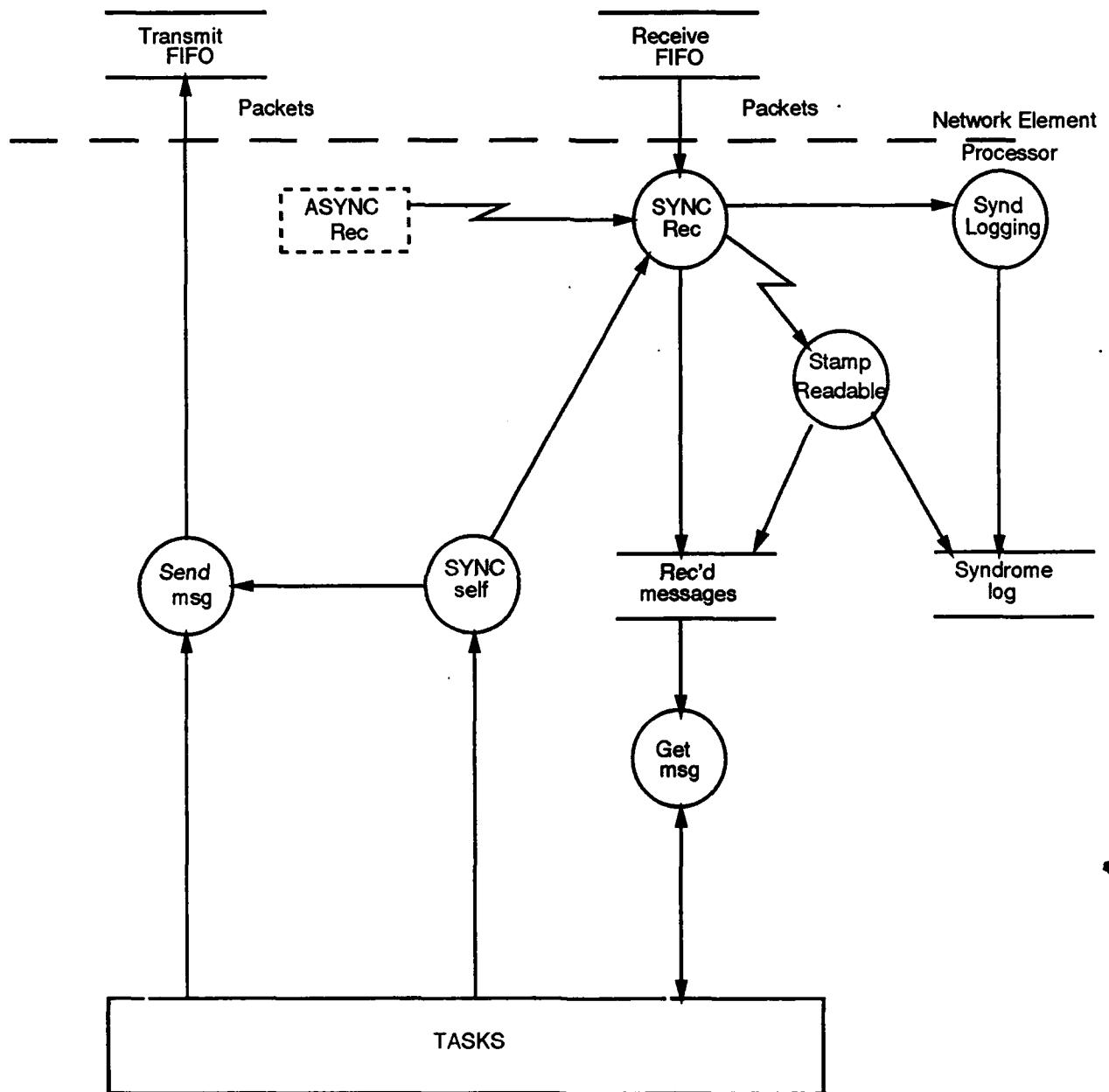


Figure 5.29. Network Services Top-Level Model

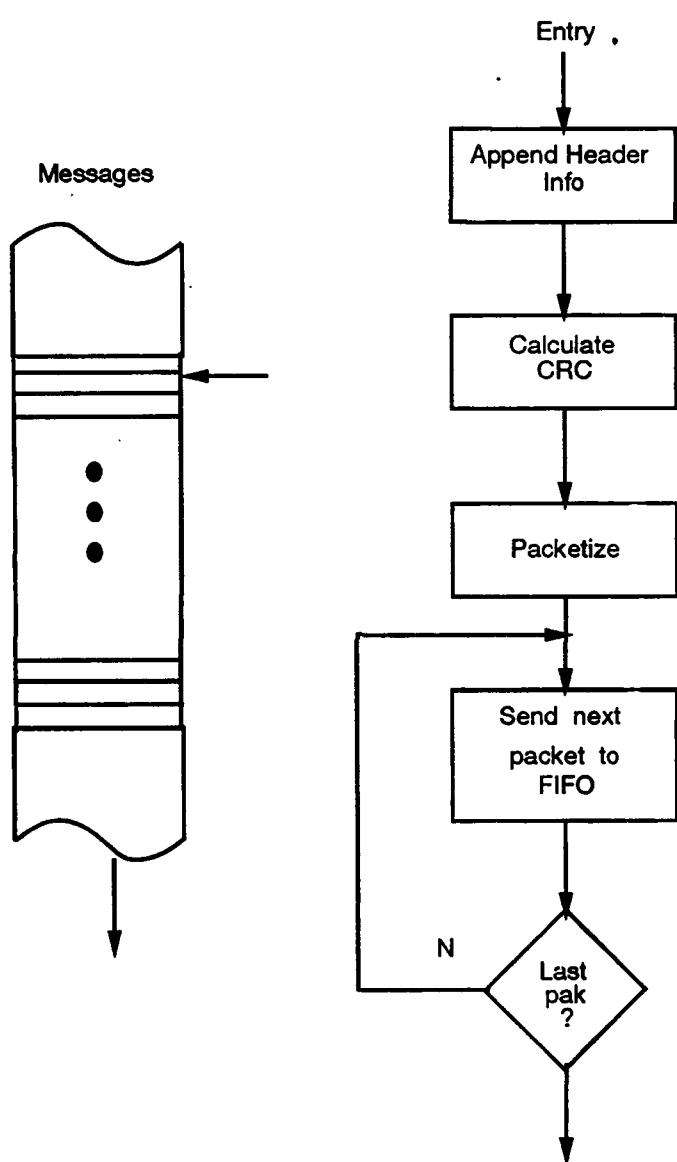


Figure 5.30. Network Services Send Message Model

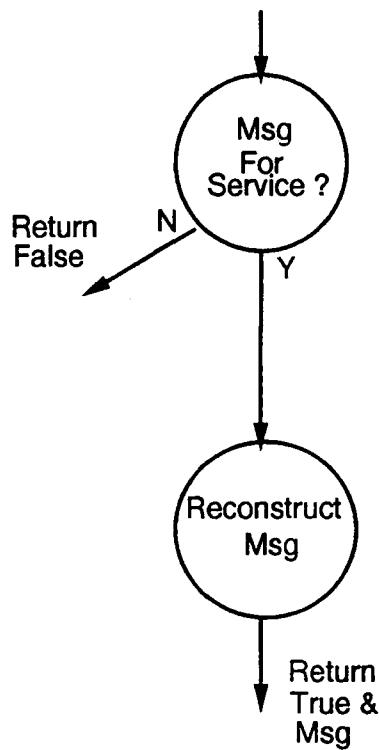


Figure 5.31. Network Services Get Message Model

syndromes is available, the FDI process uses those syndromes to determine if an error has occurred. If an error is found, all of the processors in the VID exchange their individual error vectors. If a majority of the FDI processes in the VID agree that an error has occurred, the processors exchange pertinent syndromes and attempt to identify the faulty processor. Once an identification is made, a request for reconfiguration is broadcast to all VIDs in the system. At each data exchange step, the FDI process suspends itself. It is resumed during the next scheduling loop if all expected messages have been received.

As an example of this process, consider that processor PE01 of VID1 in our 4x4 configuration is failed when VID1 transmits a message to VID0. When the message is transmitted, it is processed by network services, exchanged and voted, and a syndrome is appended. The voted message is delivered to processors PE00, PE04, and PE08, which comprise VID0, and its syndrome is logged into the syndrome log of each of the processors. To look at the FDI process in detail, consider PE00. At the next invocation of its FDI service, if there are at least 10 syndromes in the log, it will begin to process the error syndromes. After processing the syndromes, PE00 creates an error vector which indicates whether or not it has detected an error. PE00 sends the error vector to PE04 and PE08 and suspends its FDI process to await the receipt of an error vector from each of them. On the next scheduling cycle, if the messages

have been received by PE00 and stamped readable, FDI is resumed. With an error vector from each processor in the VID, PE00 can decide if one or more errors have occurred. If one or more errors have been detected, PE00 transmits the syndromes associated with each error to PE04 and to PE08 and once more suspends FDI to await the receipt of the syndromes from each of them. On the next scheduling cycle, if the syndromes have been received by PE00 and stamped readable, FDI is resumed. PE00 then identifies PE01 as the faulty component from the message class, the system configuration table, and the following syndrome information produced by this sample fault:

- Member A detected a vote error right
- Member B detected a vote error left
- Member C detected a vote error opposite
- Message class was Class 1
- Message was from VID1

After the component has been diagnosed, PE00 broadcasts a reconfiguration request message to the reconfiguration service on all VIDs and suspends its FDI service. Note that these same steps are being performed on PE04 and PE08.

The control flow model of the FDI process is illustrated in Figure 5.32.

5.2.2.2.5. Reconfiguration Models

When an error has been detected and a faulty module in a redundant processing site identified, an attempt is made to reconfigure the system to replace the faulty module with a spare module. After the FDI process that has identified the fault broadcasts a reconfiguration request, the reconfiguration services in all of the VIDs contend for reconfiguration authority (RA). The VID that wins the contention becomes the RA, notifies all VIDs that reconfiguration is in progress, and determines the new configuration that will satisfy the required redundancy levels within the constraints of separate PFCRs for each member of a redundant VID. Once the configuration has been determined, the RA broadcasts configuration table updates to all VIDS and a test message to the VID being reconfigured. It then initiates the alignment of the replacement processor with the other members of the reconfigured VID. When the alignment has been completed, the RA broadcasts a reconfiguration termination message and suspends its reconfiguration service.

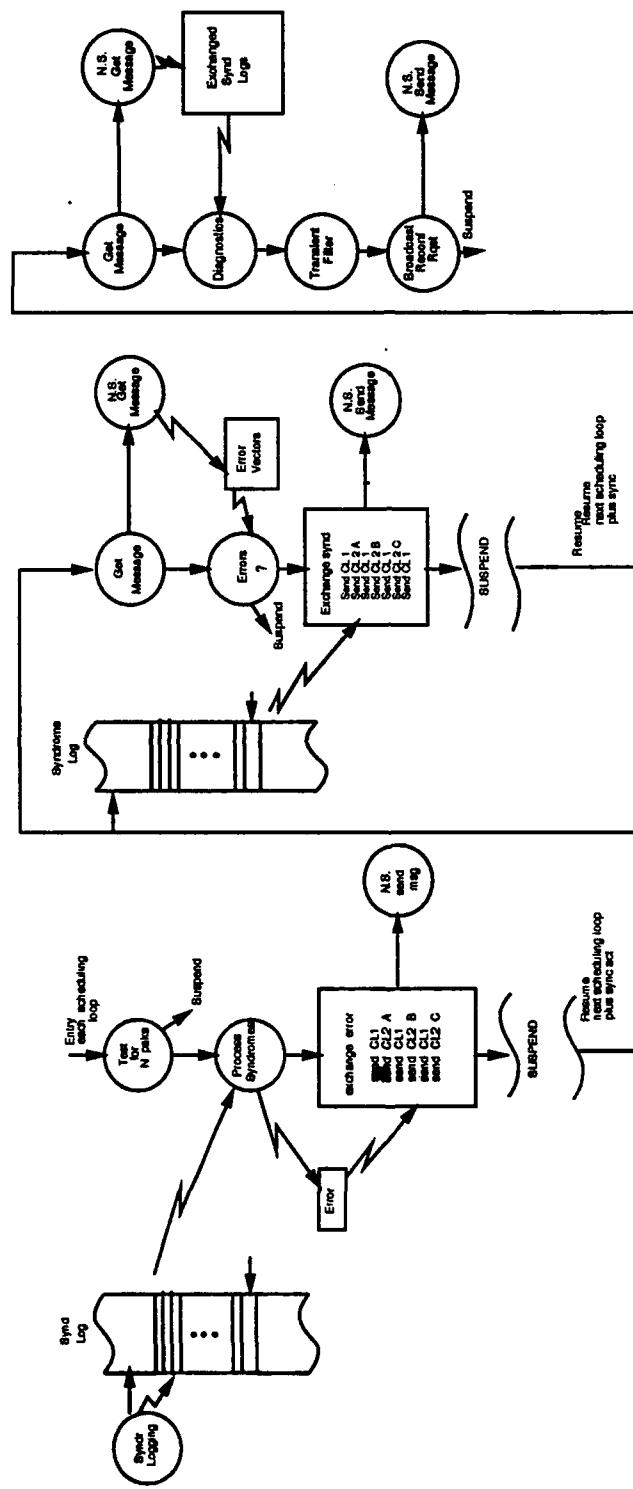


Figure 5.32. Fault Detection and Isolation Model.

The FTPP supports two different reconfiguration strategies: processor replacement and global reconfiguration. In a reconfiguration by processor replacement, a faulty processor is replaced by a simplex VID that has been designated as a spare. In global reconfiguration, the system can be completely reconfigured to contain VIDs of requested redundancy levels. The global reconfiguration strategy would be particularly useful in phased missions where differing reliability and processing requirements could result in different levels of redundancy.

5.2.2.2.5.1. Processor Replacement

The processor replacement reconfiguration algorithm consists of three basic steps:

1. Search for simplex VID in correct position for inclusion in VID undergoing reconfiguration
2. Replace faulty VID member with selected simplex VID
3. Designate faulty processor as a simplex that can resume in initial state or enter a diagnostic phase

As an example of this process, consider the FDI example described in the previous section. After VID0 detects a fault and isolates it to PE1 of VID1 and broadcasts a reconfiguration request, all VIDS of greater-than-simplex redundancy respond with a “contend for reconfiguration authority” (CFRA) broadcast to all other VIDs. In this case, VIDs 1 through 3 would contend. Assuming the CFRA from VID3 is received by the other VIDs first, VID3 then becomes the reconfiguration authority (RA). During execution of the reconfiguration algorithm, the RA examines the current configuration table (shown in Figure 5.33a) and decides to replace PE1 with PE3 since PE3 is a simplex VID and since it resides on the same network element as PE1 (and is thus in a different fault containment region than either of the other two PEs in VID1). The RA broadcasts a global notification (GN) to all VIDs that VIDs 1 and 4 are undergoing a reconfiguration and should not be sent any messages until after reconfiguration is terminated. The RA waits for GN acknowledgements from all VIDs before proceeding with reconfiguration. After receiving all GN acknowledgements, the RA broadcasts three configuration table updates (CTU). These three messages specify the three channels of VID1, replacing PE1 with PE3 for the second channel. The new VID1 is sent a test message and responds with a test acknowledgement message. The RA then broadcasts another CTU to replace PE3 with PE1 as simplex VID4. The new VID4 is sent a test message and responds with a test acknowledgement message. Finally, the RA broadcasts a reconfiguration termination message, ending

VID	Member	ME(FCR)	FIFOid	PE	VID	Member	ME(FCR)	FIFOid	PE
0	A	0	0	0	0	A	0	0	0
0	B	1	0	4	0	B	1	0	4
0	C	2	0	8	0	C	2	0	8
1	A	3	0	12	1	A	3	0	12
1	B	0	1	1	1	B	0	3	3
1	C	1	1	5	1	C	1	1	5
2	A	2	1	9	2	A	2	1	9
2	B	3	1	13	2	B	3	1	13
2	C	0	2	2	2	C	0	2	2
3	A	1	2	6	3	A	1	2	6
3	B	2	2	10	3	B	2	2	10
3	C	3	2	14	3	C	3	2	14
4	A	0	3	3	4	A	0	1	1
5	A	1	3	7	5	A	1	3	7
6	A	2	3	11	6	A	2	3	11
7	A	3	3	15	7	A	3	3	15

(a)

(b)

Figure 5.33. Processor Replacement a) Initial and b) Final Configuration Tables

the processor replacement reconfiguration. The final configuration table is shown in Figure 5.33b.

Although only a few VIDs are directly impacted by this reconfiguration, all VIDs are involved in the RA contention and reconfiguration message activity. The message types and actions that comprise the reconfiguration process are summarized in Table 5.5. A data and control flow model of activity in all PEs during reconfiguration was constructed. The message traffic that results from this model on the network elements during this reconfiguration is shown in Figure 5.34. The time of occurrence is in microseconds. Since no compute times were considered in this simulation, all compute operations occurred in zero microseconds.

5.2.2.2.5.2. Total System Reconfiguration

Like the processor replacement reconfiguration algorithm, the total system reconfiguration consists of three basic, although more complex, steps:

1. Consider current system configuration

FDI VID (0-4-8)	Rec Auth VID3 (6-10-14)	Bad PE VID1->4 (1)	Repl PE VID4->1 (3)	Repaired VID1 (5-12)	Triplex VID2 (2-9-13)	Simplexes VID 5-6-7 (7-11-15)	Time of Occur.
BrdRR							0.000
AwtRR							0.000
RcvRR	RcvRR	RcvRR	RcvRR	RcvRR	RcvRR	RcvRR	44.940
CptRR	CptRR	CptRR	CptRR	CptRR	CptRR	CptRR	44.940
BrdCFRA	BrdCFRA	BrdCFRA		BrdCFRA	BrdCFRA		44.940
AwtCFRA	AwtCFRA	AwtCFRA		AwtCFRA	AwtCFRA		44.940
RcvCFRA	RcvCFRA	RcvCFRA		RcvCFRA	RcvCFRA		153.420
CptGN							153.420
BrdGN							153.420
AwtGN	AwtGN	AwtGN	AwtGN	AwtGN	AwtGN	AwtGN	153.420
RcvGN	RcvGN	RcvGN	RcvGN	RcvGN	RcvGN	RcvGN	203.160
CptGN	CptGN	CptGN	CptGN	CptGN	CptGN	CptGN	203.160
XmtGNA	XmtGNA	XmtGNA	XmtGNA	XmtGNA	XmtGNA	XmtGNA	203.160
AwtGNA							203.160
RcvGNA							407.460
AwtCTU0	BrdCTU0	AwtCTU0	AwtCTU0	AwtCTU0	AwtCTU0	AwtCTU0	407.460
RcvCTU0	BrdCTU1	RcvCTU0	RcvCTU0	RcvCTU0	RcvCTU0	RcvCTU0	457.200
RcvCTU1	BrdCTU2	RcvCTU1	RcvCTU1	RcvCTU1	RcvCTU1	RcvCTU1	506.940
RcvCTU2	BrdScp0	RcvCTU2	RcvCTU2	RcvCTU2	RcvCTU2	RcvCTU2	556.680
RcvScp0	RcvScp0	RcvScp0	RcvScp0	RcvScp0	RcvScp0	RcvScp0	602.340
XmtTst0							602.340
			AwtTst0	AwtTst0			602.340
				RcvTst0	RcvTst0		651.660
				CptTst0	CptTst0		651.660
				XmtTAO	XmtTAO		651.660
							651.660
							701.400
							701.400
AwtTAO							
AwtCTU3	BrdCTU3	AwtCTU3	AwtCTU3	AwtCTU3	AwtCTU3	AwtCTU3	701.400
RcvCTU3	BrdScp1	RcvCTU3	RcvCTU3	RcvCTU3	RcvCTU3	RcvCTU3	751.140
RcvScp1	RcvScp1	RcvScp1	RcvScp1	RcvScp1	RcvScp1	RcvScp1	796.800
XmtTst1							796.800
							796.800
			AwtTst1				846.120
				RcvTst1			846.120
				CptTst1			846.120
				XmtTA1			846.120
							846.120
							900.235
							900.235
							900.235
AwtRT	AwtRT	AwtRT	AwtRT	AwtRT	AwtRT	AwtRT	900.235
RcvRT	RcvRT	RcvRT	RcvRT	RcvRT	RcvRT	RcvRT	949.975

Figure 5.34. Processor Replacement Message Traffic

ACTION/MESSAGE	ACRONYM	DESCRIPTION
ACTION	Brd	Broadcast message (to all VIDs)
	Xmt	Transmit message to a specific VID/PE
	Awt	Await the reception of a message
	Rcv	Receive a message
	Cpt	Compute
MESSAGE	RR	Reconfiguration Request
	CFRA	Contend for Reconfiguration Authority (RA)
	GN	Global Notification
	GNA	Global Notification Acknowledgement
	CTU	Configuration Table Update
	Scp	Scoop (Message Class 0)
	Tst	Test
	TA	Test Acknowledge
	RT	Reconfiguration Terminate

Table 5.5. Reconfiguration Messages and Actions

2. Consider requested system configuration
3. Disband and create VIDs as necessary

As an example of this process, consider the configuration shown in Figure 5.23 and assume that three faults have occurred, with the following results:

- Fault 1: PE02 fails and is replaced by PE07 as member C of triplex VID02. PE02 becomes simplex VID05.
- Fault 2: PE03, a spare simplex (VID04), fails.
- Fault 3: PE06, channel A of triplex VID03, fails.

The resulting system is shown in Figure 5.35.

As for the processor replacement reconfiguration, when the fault on PE06 is detected and isolated, a request for reconfiguration is broadcast to all VIDs. All non-simplex VIDs then contend for reconfiguration authority (CFRA), and the VID which sent the first CFRA received by all VIDs becomes the reconfiguration authority (RA). To locate a replacement PE, the RA first searches the configuration table to locate an available simplex VID. If no simplex is available, the RA searches for a VID containing one level of redundancy less than that of the disabled VID, and which can be disbanded. If no VID is available at that level, the search is repeated through all of the redundancy levels in decreasing order until an available VID is located. For the selected example, there is but one redundancy level since all redundant VIDs are triplexes. Therefore, for this example, the reconfiguration algorithm was modified to

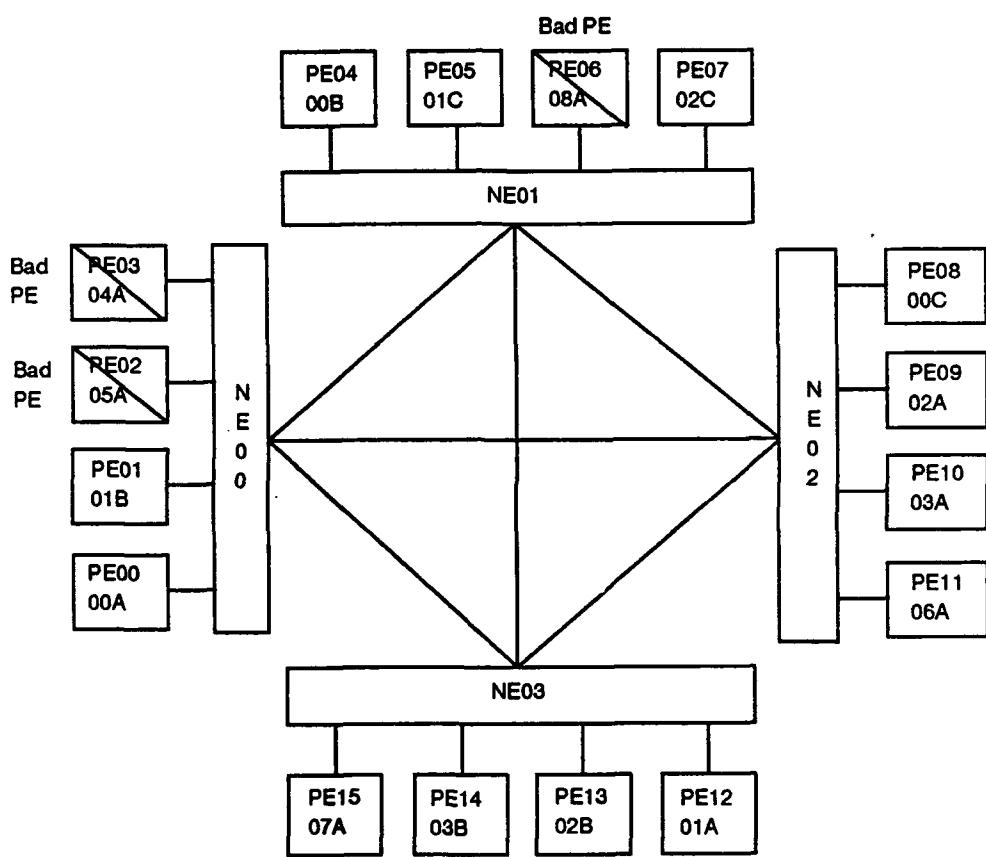


Figure 5.35. System Configuration before Total Reconfiguration

consider disbanding VIDs containing the same level of redundancy as the disabled VID.

As a result of the first two failures in the example scenario, none of the currently available simplex VIDs can be used to replace PE06 because none is in a separate PFCR from the other two members of VID03. VID02 cannot be disbanded and reconfigured because it is the RA. With the remaining VIDs, the following reconfigurations are possible:

1. VID00 can be disbanded, PE00 used to repair VID03, and PE15 used to replace PE00 in VID00
2. VID00 can be disbanded, PE04 used to repair VID03, and PE15 used to replace PE04 in VID00
3. VID01 can be disbanded, PE01 used to repair VID03, and PE11 used to replace PE01 in VID01
4. VID01 can be disbanded, PE05 used to repair VID03, and PE11 used to replace PE05 in VID01

Assuming that alternative 1 is selected, VID03 and VID00 are disbanded, creating simplex VIDs 00, 03, 08, 09, 10, and 11. This makes available seven simplex VIDs to be reassembled to satisfy the total deficit of two triplex VIDs and one simplex VID. Note that after reconfiguration, the simplex VID will be the failed processor PE06, which will be re-initialized or entered into a diagnostic phase. The following reconfiguration is chosen:

PE00: New simplex VID00 member A becomes triplex VID03 member C
PE14: New simplex VID11 member A becomes triplex VID03 member B
PE10: New simplex VID03 member A becomes triplex VID03 member A
PE08: New simplex VID10 member A becomes triplex VID00 member C
PE04: New simplex VID09 member A becomes triplex VID00 member B
PE15: Old simplex VID07 member A becomes triplex VID00 member A
PE06: Old simplex VID08 member A becomes simplex VID07 member A

The RA broadcasts a global notification (GN) to all VIDs that VIDs 00 and 03 are undergoing reconfiguration and should not be sent any messages until after reconfiguration is terminated. The RA waits for GN acknowledgements (GNAs) from all VIDS before proceeding with reconfiguration. After receiving the GNAs, the RA broadcasts the three configuration table updates (CTU) disbanding VID00 into VIDs 00, 09, and

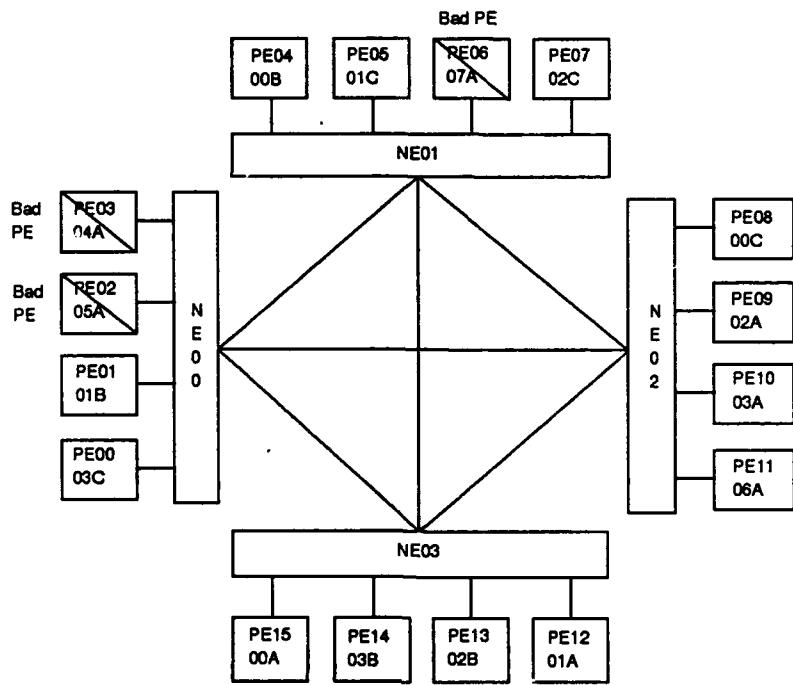
10. The RA sends test messages to VIDs 00, 09, and 10 and awaits a test acknowledgement from each VID. After receiving the three test acknowledgements, the RA broadcasts three CTUs, disbanding VID03 into VIDs 03, 08, and 011. In the same manner as for VID00, the RA sends test messages and awaits test acknowledgements. When the three test acknowledgements have been received, the RA broadcasts a GN to all VIDs that VIDs 00, 03, 07, 08, 09, 10, and 11 are undergoing reconfiguration and should not be sent any messages until after reconfiguration is terminated. Once again, the RA waits for GNAs from all VIDS before proceeding. When all GNAs have been received, the RA broadcasts three CTUs that establish triplex VID03. Then a test message is sent to VID03, which responds with a test acknowledgement. After receiving the test acknowledgement, the RA broadcasts three CTUs to establish the new triplex VID00, sends a test message to VID00, and awaits its test acknowledgement. After receiving the test acknowledgement from VID00, it repeats the CTU, test, and test acknowledgement cycle for VID07. Finally, the RA broadcasts the reconfiguration termination notice, ending the total reconfiguration process.

The final system diagram and configuration table are shown in Figure 5.36. The message traffic generated by this process is shown in Figure 5.37.

5.2.2.2.6. Architecture Model

Since the network element core provides the basic architectural support to tolerate faults, and since all activity in the system is regulated by the processing of messages in the network element core, the architecture model needs to capture the functionality and behavior of the network elements to the extent that message content and the results of network activity can be observed and manipulated. In addition, the network elements have to be modeled to a degree of resolution sufficient to trigger the FDIR mechanisms and allow their functioning to be simulated. Therefore, a network traffic, or message transport, model was created as the core of the system model. The network traffic model simulates the flow of data through the network elements of the selected FTTPP configuration, including the recognition of message type, the message processing appropriate for each type, the voting and error syndrome process, and the delivery of messages to the indicated destination. The network communication frame consists of a local exchange request pattern (LERP), a system exchange request pattern (SERP), scoreboarding, honoring the requests, and a synchronization pattern. This frame is illustrated in Figure 5.38.

The inputs to the model are the configuration table, which controls the assignment of a processor on a particular NE FIFO to a particular VID, and a file for each processor containing message traffic to be transmitted by that PE. Each entry in the message transmission file contains the time for the message to be transmitted, the message type



VID	Member	NE(FCR)	FIFOid	PE
03	C	00	0	00
03	B	03	2	14
03	A	02	2	10
02	C	01	3	07
02	B	03	1	13
02	A	02	1	09
01	C	01	1	05
01	B	00	1	01
01	A	03	0	12
00	C	02	0	08
00	B	01	0	04
00	A	03	3	15
07	A	01	2	06 (3)Bad PE:3A/3B/3C->8A/3A/3B
06	A	02	3	11
05	A	00	2	02 (1)Bad PE:2C/5A->5A/2C
04	A	00	3	03 (2)Bad PE

Figure 5.36. Final a) System Diagram and b) Configuration Table

FDI VID	Rec Auth	Duplex	Triplex	Simplexes	Time
VID00	VID02	VID03	VID00/03	VID06/07	
00-04-08	09-13-07	10-14	10-14-00	06-15	
			15-04-08	02-03-11	
			01-05-12		
					000.000
					000.000
BrdRR	RcvRR	RcvRR	RcvRR	RcvRR	044.940
AwtRR	CptRR	CptRR	CptRR	CptRR	044.940
BrdCFRA	BrdCFRA	BrdCFRA	BrdCFRA		044.940
AwtCFRA	AwtCFRA	AwtCFRA	AwtCFRA		044.940
RcvCFRA	RcvCFRA	RcvCFRA	RcvCFRA		153.420
			CptGMO		153.420
			BrdGMO		153.420
AwtGMO	AwtGMO	AwtGMO	AwtGMO	AwtGMO	153.420
RcvGMO	RcvGMO	RcvGMO	RcvGMO	RcvGMO	203.160
CptGMO	CptGMO	CptGMO	CptGMO	CptGMO	203.160
XmtGMAO	XmtGMAO	XmtGMAO	XmtGMAO	XmtGMAO	203.160
			AwtGMAO		203.160
			RcvGMAO		392.380
AwtCTUO	BrdCTUO	AwtCTUO	AwtCTUO	AwtCTUO	392.380
RcvCTUO	BrdScpO	RcvCTUO	RcvCTUO	RcvCTUO	442.120
RcvScpO	RcvScpO	RcvScpO	RcvScpO	RcvScpO	487.780
	XmtTst0				487.780
AwtTst0	<-----				487.780 VID00A
RcvTst0					537.100 .
CptTst0					537.100 .
XmtTAO					537.100 (PE00)
	-> AwtTAO				537.100
	RcvTAO				591.215
	XmtTst1				591.215
AwtTst1	<-----				591.215 VID09A
RcvTst1					640.955 .
CptTst1					640.955 .
XmtTA1					640.955 (PE04)
	-> AwtTA1				640.955
	RcvTA1				695.070
FDI VID	Rec Auth	Duplex	Triplex	Simplexes	Time
VID00	VID02	VID03	VID00/03	VID06/07	
00-04-08	09-13-07	10-14	10-14-00	06-15	
			15-04-08	02-03-11	
			01-05-12		
					695.070
					695.070 VID10A
XmtTst2	<-----				744.810 .
AwtTst2					744.810 .
RcvTst2					744.810 .
CptTst2					744.810 .
	-> AwtTA2				744.810
	RcvTA2				798.925
AwtCTU1	BrdCTU1	AwtCTU1	AwtCTU1	AwtCTU1	798.925
RcvCTU1	BrdScp1	RcvCTU1	RcvCTU1	RcvCTU1	848.665
RcvScp1	RcvScp1	RcvScp1	RcvScp1	RcvScp1	894.325
	XmtTst3				894.325
	----AwtTst3				894.325 VID03A
	RcvTst3				943.645 .
	CptTst3				943.645 .
	XmtTA3				943.645 (PE10)
AwtTA3	<-----				943.645
RcvTA3					997.760
XmtTst4					997.760

Figure 5.37. Total Reconfiguration Network Element Message Traffic

					997.760 VID11A
		AwtTst4			1047.080 .
		RcvTst4			1047.080 .
		CptTst4			1047.080 (PE14)
		XmtTA4			
		AwtTA4<-----			1047.080
		RcvTA4			1101.195
		BrdGHI1			1101.195
	AwtGHI1	AwtGHI1	AwtGHI1	AwtGHI1	1101.195
	RcvGHI1	RcvGHI1	RcvGHI1	RcvGHI1	1150.935
	CptGHI1	CptGHI1	CptGHI1	CptGHI1	1150.935
	XmtGHA1	XmtGHA1	XmtGHA1	XmtGHA1	1150.935
	AwtGHA1				1150.935
	RcvGHA1				1229.675
FDI VID	Rec Auth	Duplex	Triplex	Simplexes	Time
VID00	VID02	VID03	VID00/03	VID06/07	
00-04-08	09-13-07	10-14	10-14-00	06-15	
			15-04-08	02-03-11	
				01-05-12	
-----	-----	-----	-----	-----	-----
AwtCTU2	BrdCTU2	AwtCTU2	AwtCTU2	AwtCTU2	1229.675
RcvCTU2	BrdCTU3	RcvCTU2	RcvCTU2	RcvCTU2	1274.495
RcvCTU3	BrdCTU4	RcvCTU3	RcvCTU3	RcvCTU3	1319.315
RcvCTU4	BrdScp2	RcvCTU4	RcvCTU4	RcvCTU4	1364.135
RcvScp2	RcvScp2	RcvScp2	RcvScp2	RcvScp2	1404.875
XmtTst5					1404.875
	----->AwtTst5				1404.875 VID03A
	RcvTst5				1449.275 VID03B
	CptTst5				1449.275 VID03C
	XmtTA5				1449.275 (PEs10-14-00)
	AwtTA5<-----				1449.275
	RcvTA5				1503.390
AwtCTU5	BrdCTU5	AwtCTU5	AwtCTU5	AwtCTU5	1503.390
RcvCTU5	BrdCTU6	RcvCTU5	RcvCTU5	RcvCTU5	1554.415
RcvCTU6	BrdCTU7	RcvCTU6	RcvCTU6	RcvCTU6	1605.440
RcvCTU7	BrdScp3	RcvCTU7	RcvCTU7	RcvCTU7	1656.465
RcvScp3	RcvScp3	RcvScp3	RcvScp3	RcvScp3	1702.125
XmtTst6					1702.125
	----->AwtTst6				1702.125 VID00A
	RcvTst6				1746.525 VID00B
	CptTst6				1746.525 VID00C
	XmtTA6				1746.525 (PEs15-04-08)
	AwtTA6<-----				1746.525
	RcvTA6				1800.640
AwtCTU8	BrdCTU8	AwtCTU8	AwtCTU8	AwtCTU8	1800.640
RcvCTU8	BrdScp4	RcvCTU8	RcvCTU8	RcvCTU8	1851.665
RcvScp4	RcvScp4	RcvScp4	RcvScp4	RcvScp4	1897.325
XmtTst7					1897.325
	----->AwtTst7				1897.325 VID07A
	RcvTst7				1941.725 .
	CptTst7				1941.725 .
	XmtTA7				1941.725 (PE06)
	AwtTA7<-----				1941.725
	RcvTA7				1995.840
	BrdRT				1995.840
AwtRT	AwtRT	AwtRT	AwtRT	AwtRT	1995.840
RcvRT	RcvRT	RcvRT	RcvRT	RcvRT	2045.580

Figure 5.37. Total Reconfiguration Network Element Message
(Continued)

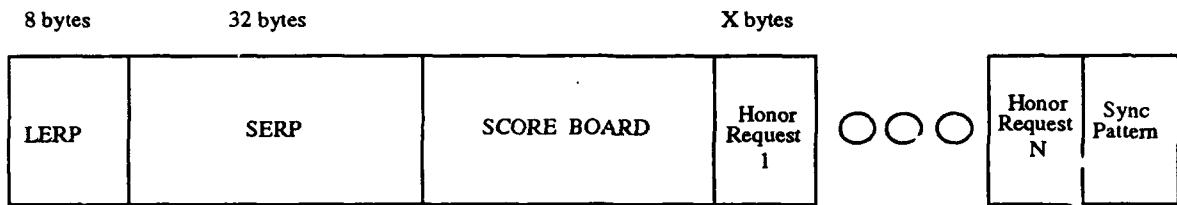


Figure 5.38. Network Communication Frame

(or exchange class), the destination VID, and the content of the message. The output from the model is a file of message traffic, partitioned into transmitted messages and received messages. For the transmitted messages, the actual time of transmission and the message content are listed. For the received messages, the time of receipt, the voted message content, and the error syndrome appended by the voter are listed.

This FTPP network traffic, or message transport, model was initially implemented as a functional data-flow model using the ADAS functional simulator CSIM. This implementation consists of hierarchical graphs and "C" code attached to graph nodes and arcs to describe the functionality of the system component represented by the node or arc. The model consisted of 13 distinct graphs containing 207 nodes and 274 arcs in its unexpanded form with 45 distinct "C" modules. The simulation of the model resulted in a 7.22 million-to-one slow-down.

The CSIM network traffic model was used with preliminary application and reconfiguration models to establish some of the baseline timing information for network processes. For example, using this model and the application model described above, the network element timing data contained in Table 5.6 was computed.

5.2.2.2.7. Integrated Model

The evaluation of the performance and effectiveness of the fault tolerance features of an architecture is the main goal of the design refinement modeling phase. This evaluation requires the integration of a number of models of system components into a system behavioral model that can simulate the actual processing of data at a functional level and that can simulate faulted and fault-free operation. Since a balance has to be achieved between the level of detail captured by the model and the time required to simulate it, the integrated system model was constructed around a core behavioral model of network data flow and functional models of processing element activities. Models of the operating system processes, the fault detection and identification processes, the reconfiguration processes, and an application algorithm were integrated into the system model as services of the processing element models.

Table 5.6. Total Network Element Traffic for Fork and Join Application

Network Frame Time	PE Data Schdld. Time	PE Data Actual Time	PE Data Receive Time	Delta Time	Message Class	Origin VID	Dest. VID	Network Frame Time	PE Data Schdld. Time	PE Data Actual Time	PE Data Receive Time	Delta Time	Message Class	Origin VID	Dest. VID
0.00								362.10	375	427.04	52.04	1	0	0	0
15.08	25	25	75.1	50.1	1	0	0	381.68	397.18	400	446.62	46.62	1	1	1
30.16	50	50	99.055	49.055	2c	0	0	416.34	425	466.2	41.2	1	2	2	2
45.24	75	75.1	121.4	46.3	1	0	0	435.92	450	450	500.86	50.86	1	3	3
64.40	100	100	157.67	57.67	2c	0	0	455.50	471.00	475	520.44	45.44	1	0	0
88.36	125	125	179.07	54.07	1	0	0	490.18	500	500	555.1	55.1	1	1	1
107.94	150	157.67	221.4	63.73	2c	0	0	509.74	525	525	574.68	49.68	1	2	2
123.44	175	179.07	246.4	67.33	1	0	0	525.24	544.40	550	594.26	44.26	1	3	3
146.97	200	200	259.82	59.82	2c	0	0	563.98	575	575	628.92	53.92	1	1	0
166.55	225	225	279.4	54.4	1	0	1	583.56	599.06	600	648.5	48.5	1	2	0
182.05	250	259.82	314.06	54.24	1	0	1	618.22	625	625	668.08	43.08	1	3	0
205.59	275	275	333.64	58.64	1	0	2	637.80	657.38	672.88					
225.17	300	300	353.22	53.22	1	0	2								
249.12	325	325	372.8	47.8	1	0	3								
268.70	350	350	392.38	42.38	1	0	3								
284.20															
303.36															
322.94															
342.52															

To demonstrate the modeling capability that is needed to construct this integrated model and to provide the flexibility of including different models of various processes for different simulation experiments, an object-based discrete-event simulator of the FTPP was constructed. In this model, autonomous components in the system are represented by modeling objects. The interaction of the autonomous components are represented by the interactions of the modeling objects. For each modeling object, a set of known interaction points are defined, and all interactions with that object take place at these interaction points. The FTPP simulator was embedded in a simulation harness that allowed the collection of simulation data for the computation of performance measures. The simulator was written in Ada to take advantage of the Ada tasking facilities for the construction of an object-based program. Each system component was modeled by an Ada task and their interactions were modeled by task rendezvous under the control of a system schedule task.

The core modeling object of the system is the scheduler object. This scheduler supports the concept of simulated time. Other model objects interact with the scheduler to insert time delays into their interaction sequences with other modeling objects. When no other model interactions can take place, the scheduler object allows time to advance by the shortest delay expected by the model objects that are interacting with it. Simulated time advances at an uneven rate as determined by the other model objects in the modeling system. The simulated time will, however, model the exact real time imposed by the simulated constraints of the real system.

The application, operating system, fault detection and isolation, and reconfiguration algorithms are modeled as service objects defined for the processing element objects. These objects then interact with the network element objects under the constraints imposed by the scheduler. With this model, the network data flow generated by the services of each processor in the sample configuration can be modeled. This allows the operating system impact on processor and network element activity to be simulated and allows the application model to interact with the fault detection and isolation and the reconfiguration processes. As a consequence of using an Ada-based modeling technique, these algorithms can be expressed directly in the models as they would be implemented on the target architecture.

The inputs to the model are the initial configuration table, the system processes, the fault injection scenario files, and the application processes. The system processes that are currently modeled are fault detection and isolation, reconfiguration, timekeeper, and fault-masking-group synchronization. The currently supported fault injection scenarios are data-link parity errors, transmit and receive FIFO data errors, voter errors, voter syndrome errors, and reflect FIFO errors. The outputs from the model are tracing data for each PE for network traffic and application activity and utilization information for network components, operating system functions, and application times for each process. The tracing data for the network traffic includes time stamps

for packet transmit, packet receive, packet marked readable, and packet read. The tracing data for the application activity includes time stamps for execution, message transmit, and message receive. The utilization information for operating system functions includes context switch time, message processing, and time spent awaiting message receipt.

The model consists of approximately 300 Ada tasks in 31 packages. There are approximately 800 lines of code in the simulator harness; 3,300 lines of code for network data flow; and 5,000 lines of non-network code. The simulator performs at a 450,000-to-one slowdown on a general-purpose computer of approximately 1 MIPS throughput. Figures 5.39 through 5.41 contain Buhr charts [30] illustrating the structure of the model.

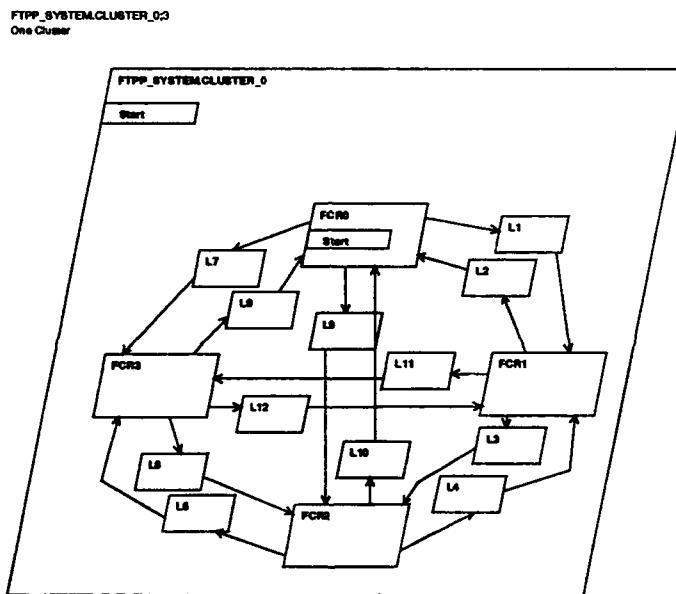


Figure 5.39. Cluster Level of FTPP Model

5.2.2.3. Results

Following the construction and test of the integrated model, simulation experiments or test cases were defined to evaluate various parts of the FTPP design. One set of experiments was set up and carried out using the fault-free integrated system model. Experiments and results analyses examined 1) the message transport functionality and performance, 2) the system configuration control logic functionality, 3) the network element synchronization, 4) the scheduler functionality and performance, 5) the network services functionality and performance, 6) the application process

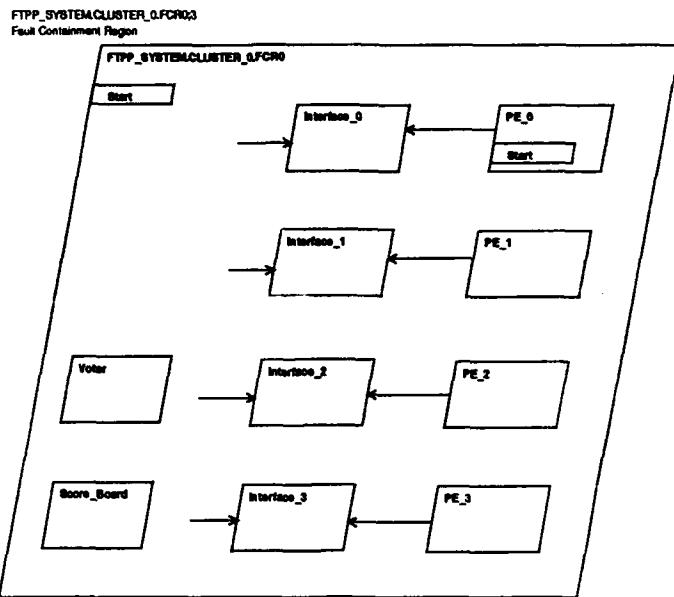


Figure 5.40. Network Element/FCR Level of FTPP Model

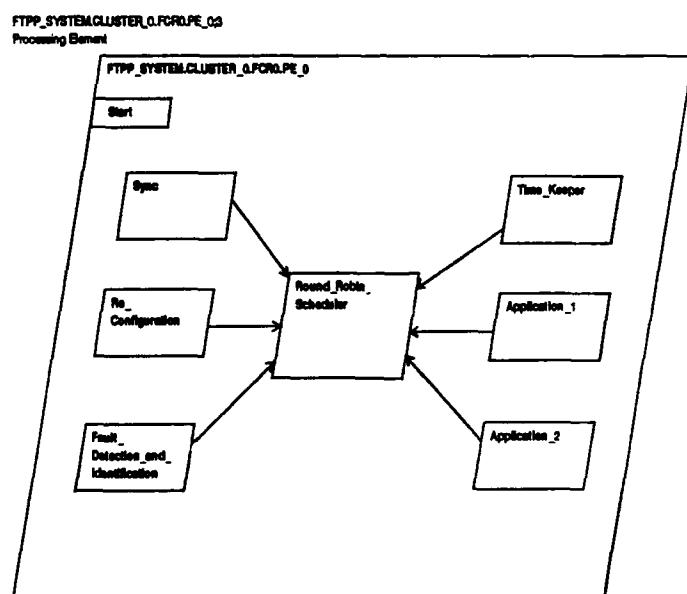


Figure 5.41. Processing Element Level of FTPP Model

functionality and performance, and 7) the task synchronization mechanism functionality.

Experiments were also set up and carried out using the integrated system model under various faulted conditions. Analyses of results from the experiments examined 1) the functionality of network element fault detection and reporting, 2) the functionality of the fault detection and identification process, 3) the response time of the fault detection and identification process, 4) the functionality of the reconfiguration process, 5) the correctness of the configuration selected by the reconfiguration process, 6) the response time of the reconfiguration process, 7) the adequacy of the state alignment process for reconfigured units, 8) the effect of a faulty processor on system throughput prior to the reconfiguration process, and 9) the assessment of fault containment. The fault conditions included 1) transmit FIFO data errors, 2) receive FIFO data errors, 3) voter errors, 4) voter syndrome errors, and 5) reflect FIFO errors.

The first test of the integrated model under faulted conditions involved corrupting the transmit FIFO data associated with one processor in one triplex fault-masking group. As a result of imposing this fault, the two non-faulted members of the triplex fault-masking group failed to respond as expected. A careful and thorough examination of the model's behavior was conducted. It was determined that the model properly reflected the design descriptions used to implement the model. Furthermore, it was determined that the FTPP design as represented by those descriptions did, in fact, contain a design flaw. The flaw would, under certain conditions, allow a faulty processor to induce faulty behavior in the functioning processors of a fault-masking group. Therefore, the fundamental property of fault containment upon which fault tolerance depends was not satisfied.

The problem stemmed from the handling of message task destination addresses for Class 2 messages containing information possessed by a single processor within a fault masking group that is to be distributed to the other processors. For this experiment, fault containment failed during the transmission of a Class 2 message from a processor whose transmit FIFO was faulted. This Class 2 message transmission contained error vectors from within the faulted processor's FDI task to the companion FDI tasks in the other two processors of the fault masking group. The task destination address within the message was corrupted by the injected fault. Since a Class 2 message was being sent, there was only one source for the task destination address and it was corrupted. The corrupted address caused the message to be delivered to application tasks in the receiving processor rather than the desired FDI tasks. Consequently, the FDI tasks in the non-faulty processors continued to wait for receipt of the expected Class 2 message and the application tasks received a message intended for the FDI. Figure 5.42 diagrams this logical fault-containment problem.

Discussions with the chief architect for the FTPP at Draper Laboratory indicated

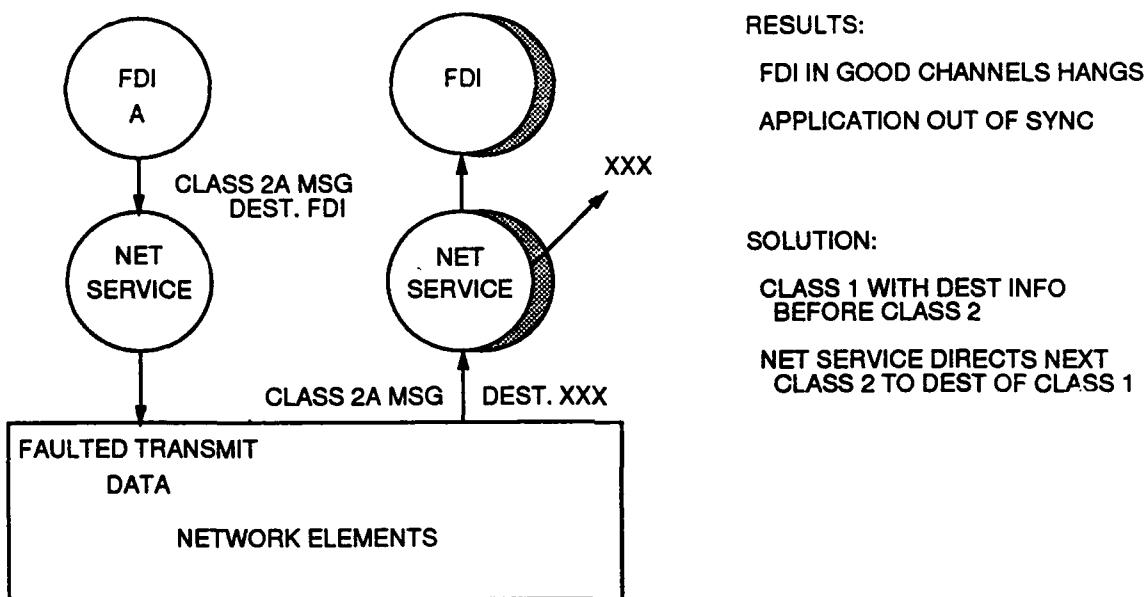


Figure 5.42. Logical Fault Containment Design Error

that this particular problem did, in fact, exist in an early prototype of the FTPP and had been found during development of the FDI software function. The problem was corrected at that time, but the documentation used for this task did not include that correction. This problem was corrected by using a Class 1 message to transmit the task destination information prior to the Class 2 message. Network communication services software was modified to use the destination information from this Class 1 message to direct the next arrival Class 2 message to the proper task. Since the destination task information is sent via Class 1 (that is, all functioning processors in the fault masking group send the correct destination information), erroneous destination information from a faulty processor will not control the delivery of the Class 2 message within the functioning processors. This correction was also implemented in the integrated model before conducting further experiments.

Experiments were conducted using the integrated FTPP behavior model under faulted conditions to determine fault detection, identification, and reconfiguration (FDIR) time values. Figure 5.43 diagrams the sequence of events associated with one of these experiments. VID1 and VID2 in this scenario are independent triplex processing sites, VID5 is a simplex that can be used as a spare. In this experiment, a fault was injected in PE0, channel A of VID1, at time zero. The sequence of events in the figure represents the activities and their time of occurrence as the fault is detected and isolated, a reconfiguration authority (VID2) is selected, and PE3 (VID5) replaces the faulty PE0. Fault detection and recovery time values determined by these experiments were then input into a reliability model. The reliability model and analysis based on these time values are discussed in Section 5.2.3 of this report.

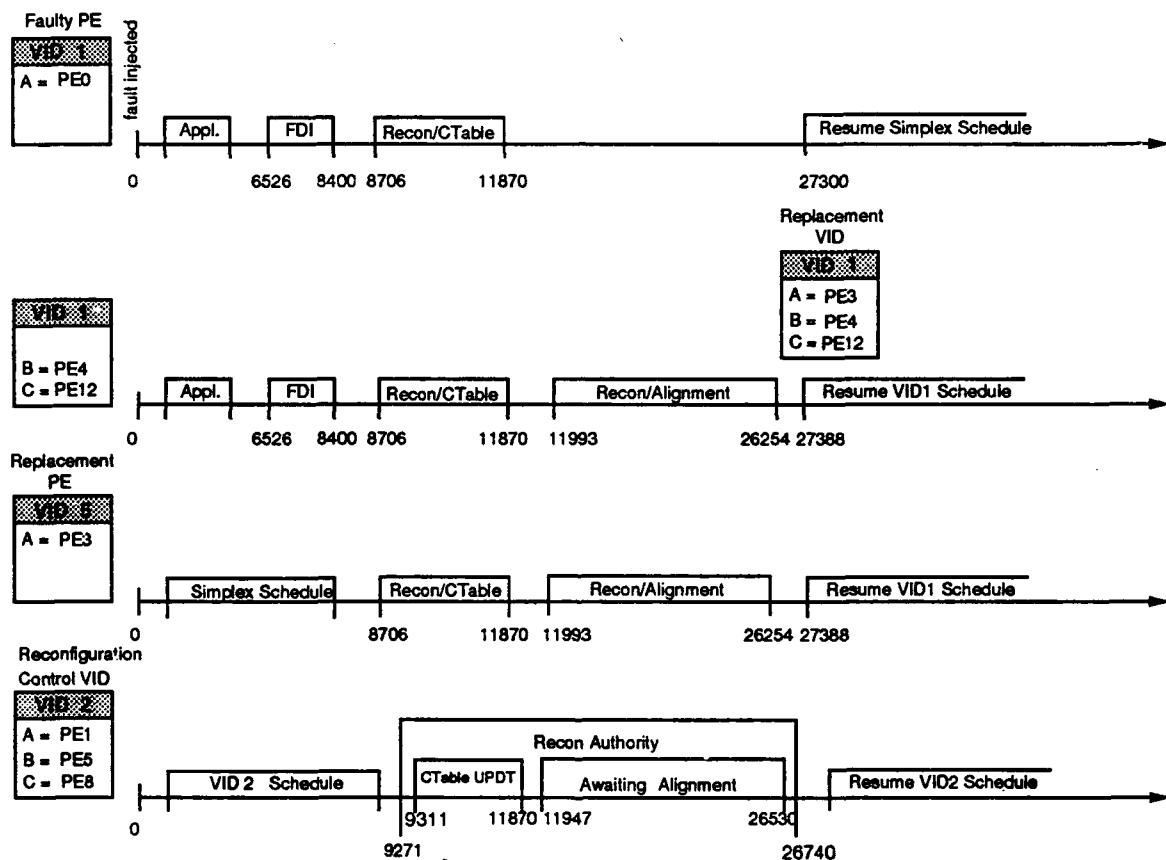


Figure 5.43. Faulted Run Time Line

In addition to determining functionality failure effects, fault coverage, and fault recovery times, the behavior model can be used to support performance evaluation. Since these behavior models are developed at a later stage and incorporate more detailed design information than is used in high-level performance models, performance evaluation based on these models can produce higher fidelity results.

One of the primary performance concerns for a parallel processing application is the mechanisms used to provide communications between processors. Since the functionality of the message transport architecture and supporting network communication services software was represented in this behavior model, the communication system performance can be predicted by this model. Data from process simulations using this model indicate that FTPP communications bandwidth is limited by the network communication services software. The message delivery time associated with this software is many (30-60) times the time required to transmit the message on the physical communication links. Consequently, the prototype FTPP communications performance could not be substantially improved by simply increasing the bandwidth of the communication links.

5.2.2.4. Conclusions

The results of this effort demonstrate the potential value of behavioral modeling in support of fault tolerance evaluation prior to implementation. It is important to note that the results of this case study apply only to the prototype of the FTPP on which it was based and not to subsequent versions. The prototype was selected for this case study so that design information consistent with early- to mid-development phases could be used in building a model to evaluate the effectiveness of fault tolerance mechanisms. Fault injection experiments using this model were carried out and the results led to the identification of a fault containment problem and to the determination of key FDIR parameters for use in reliability analysis.

In this case study, a potentially serious design flaw in a fault-tolerant system was identified using models built from design information consistent with the early- to mid-design stages. Although this problem had already been found and corrected by Draper Laboratory while developing FDI software for the prototype, the independent discovery of this design flaw through simulation is significant. While this particular problem could be resolved by a software change, the solution was obtained at the expense of a modest performance penalty and increased functional complexity in the network communication services software. A hardware solution which would not require a performance loss and increased software complexity was precluded by the cost of redesigning the prototype network element hardware. It is fortunate that this particular design problem could be resolved with only a modest loss in performance.

In general, there is no guarantee that a solution of moderate cost could always be found. The use of models early in the design to verify design concepts can reduce the likelihood of design flaws persisting into implementation. Flaws can then be corrected before it is too late or too costly to determine and implement the best solution.

In addition to these results, lessons were learned regarding the value, limitations, and automation requirements for simulation-based fault tolerance evaluation. In general, it is concluded that behavior models can provide useful support for both the design and evaluation of fault tolerance mechanisms. One of the difficult problems in replacing a failed processor in a fault-masking group with a spare is determining the specific state information required to align the spare processor with the remaining processors in the fault masking group. In carrying out this case study, it was found that alignment stands out as one of the primary areas where these models can be used. For this case study, the state information required for proper alignment had not been specified for FTPP applications. To model the reconfiguration process, it was necessary to determine the state information and steps required to achieve an acceptable alignment of a replacement processor. The simulation model was used in an iterative process in which the reconfiguration process model was modified and tested to determine if alignment could be reached. This process led to identification of all state information required for alignment for the application used for this case study. Acceptable state alignment for this simple fork-and-join application required not only application state variables but also state variables contained in other tasks, such as the message buffers in the network services task. Use of behavior models appears to be an effective way to address alignment issues early in the development process and could result in the identification of serious alignment problems for certain applications or architectures.

One of the major problems encountered in carrying out this case study was management and effective analysis of the model outputs from experiments. The detailed event sequence for simulation intervals of a few hundred milliseconds and for the 16 processors produce very large information files. To effectively analyze these results, rudimentary tools were built to sort through, select, and consolidate the information of interest from a particular run. The capability to find specific event conditions and to produce simplified timing sequence diagrams was also implemented. For larger experiments or for systems requiring greater numbers of processors (such as would be required for SDI applications), the management and analysis of simulation outputs represents a major problem. Tools developed to support this overall methodology must be capable of managing and analyzing these large experiment outputs so that the user can effectively carry out evaluations and make design decisions.

In simulation-based experimental evaluation of fault-tolerant systems, the model is a surrogate for the real system which allows faults and/or errors to be injected. In the model, the response to each fault (delays, degradation, recovery) can be characterized.

The advantages of this evaluation is that it provides access to fault injection points not necessarily accessible in the real system and allows an evaluation of the system at points in the design cycle when the real system may not be available. The limitations of this evaluation are that it may be difficult to attain a high level of model fidelity and that computing for a large fault set is not feasible.

If it is assumed that N is the number of components or functions in a system that is being simulated, that the computer workload to simulate a system is proportional to N , that the number of single faults that can be considered is also proportional to N , and that N test scenarios are required to test the system, then the fault experiment space grows as N^3 . The number of scenarios or sequences required to test the system depends upon the particular system. Simple functions can in some instances be tested in the assumed N test scenarios, but other cases may require K^N scenarios.

The simulation time for the FTPP behavior model used in this case study was approximately 500,000 times greater than the time interval being simulated. For moderately complex systems, exhaustive testing via simulation is not feasible and is not being recommended. However, simulation can play an important role in guiding design decisions, examining selected areas of system behavior, and in identifying problem areas. Techniques for determining the fault space to be examined by simulation need to be developed.

5.2.3. Reliability Analysis with Measured Parameters

5.2.3.1. Background

The reliability models created in the initial design modeling phase must be refined and expanded for the level of modeling required of the design refinement phase. The initial design phase models are constructed based on initial assumptions regarding the fault detection, isolation, and recovery (FDIR) processes that are proposed for the architecture. Also, no empirical data is available for estimating transition rates in the model related to those processes. As a result of that modeling, however, estimated bounds on the performance and effectiveness that would be required of the FDIR processes to meet the reliability requirements can be made. As the design proceeds and the FDIR processes are further defined and developed, the functional/behavioral fault tolerance performance models described for the design refinement modeling phase can be used to verify that the reliability model assumptions about those processes are correct and that the necessary bounds predicted by the model can be attained.

In the case study described in Section 4.2.3, a reliability model of a Fault-Tolerant Parallel Processor (FTPP) configuration consisting of four network elements (NEs) and four processing elements (PEs) per network element was constructed and analyzed. In the case study described in Section 5.2.2, a fault tolerance performance model of the FTPP was constructed and analyzed for the same configuration. The case study described in this section uses the reliability model and the results of the fault tolerance performance modeling to illustrate how performance modeling results can be used to refine and verify reliability models. A first-level refinement of the reliability model was accomplished by creating a detailed model of FTPP detection and recovery states for permanent faults. Measured data from the fault tolerance performance modeling of the FTPP FDIR process under faulted conditions were then used in this detailed model. Also, the phased mission model described in Section 4.2.3.3.3 was recomputed using this measured data.

5.2.3.2. Description

The initial design phase reliability model of the FTPP was constructed to contain fault recovery transitions, but no states and transitions to model the behavior of the recovery process were included. This lack of detail corresponds to the level of information that would be available for the FDIR processes at this early stage in the design. One of the FDIR assumptions underlying the model is that the system can be reconfigured to use any spare processing element to replace any failed processing element.

Since no two processors in a redundant processing, or fault-masking, group (FMG) can be attached to the same network element, any spare cannot necessarily replace any failed processor. However, based on initial descriptions of the two reconfiguration strategies, it seemed plausible to assume that FMGs could be reorganized as necessary to use the available spares. As more detailed descriptions of the reconfiguration strategy became available, this assumption was found to be overly optimistic for the selected configuration. The reliability predictions from the model were computed for assumed ranges of recovery rates and effectiveness. These ranges were based on reasonable (e.g., past experience) estimates, not empirical data. Therefore, in the design refinement phase, the reliability model was refined to allow a more detailed look at the recovery process and to use empirical data from the performance modeling to verify and/or correct its assumptions and predictions.

Since the existing model of the FTPP from the initial design phase case study borders on being too complex to be computed within a reasonable length of time for certain parameter ranges and mission times, the FTPP recovery model was analyzed for only one triplex in this case study. The FTPP recovery model for one triplex and one spare is illustrated in Figure 5.44. In the starting state, labeled (3,1), there are three

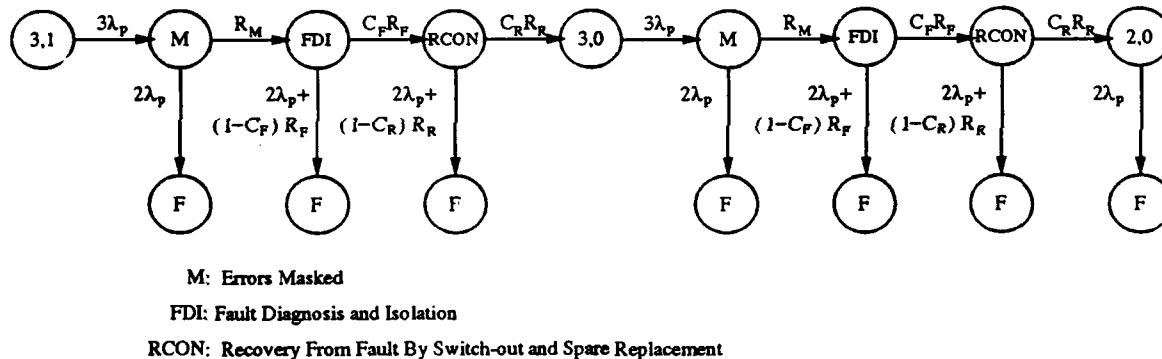


Figure 5.44. Triplex with FTPP Detection and Recovery States

active processors and one spare. When a fault occurs on any of the three processors, the system enters the state labeled "M," where errors produced by the fault are masked. When a required minimum level of message traffic through the system has been completed, the fault detection and isolation process is triggered, and the system enters the state labeled "FDI." The length of time spent in M is denoted by R_M . If the fault detection and isolation process can determine that an error has occurred and identify a faulty processor, the reconfiguration process is triggered, and the system enters the state labeled "RCON." The length of time required to detect and isolate a fault and request reconfiguration is denoted by R_F . The reconfiguration process takes the faulty processor off line, locates a spare, aligns the spare with the two remaining processors, and the system enters the state labeled (3,0) with three active processors and zero spares. The time to reconfigure is denoted by R_R . From the (3,0) state,

the process repeats itself at the next fault occurrence. However, if a second fault should occur while the system is in any of the M, FDI, or RCON states, the system is assumed to fail. The transition from the FDI to the RCON state is conditioned by the probability C_F that the FDI process can detect the error and identify the faulty processor. C_F is thus a measure of the effectiveness of the FDI process and R_F is a measure of its performance. With probability $(1 - C_F)$, FDI is unsuccessful, and the system fails. Likewise, the transition from RCON to (3,0) is conditioned by the probability C_R that reconfiguration is successful.

To model the determination of whether or not an existing spare is available as a replacement for a particular processor, the RCON state would have to be expanded to branch into the two different reconfiguration processes, or the conditioning probability C_R could be adjusted to reflect the likelihood of successful replacement at a particular system state. An additional refinement would be the inclusion of states to model the transient analysis and recovery process.

5.2.3.3. Results

Simulations of the FTPP fault tolerance performance model were conducted in case study 5.2.2, with a fork-and-join application program executing on four parallel processing sites. Each processing site consisted of one triplex. One triplex was designated for the master task, which collected data, distributed it to the other sites, computed a portion of the task, and collected results from the other processing sites. The other three processing sites were designated for the slave tasks of receiving data, computing, and returning data to the master task. Faults were injected in the transmit and receive FIFO's of channels in both the master and slave sites and in the voting mechanisms in the model, and the model was simulated. From these simulations, the times R_M , R_F , and R_R were computed. The reliability model was then instantiated with these values and solved. The reliability model was also solved for different values of C_F and C_R .

The case summarized in Table 5.7 considered different values for R_M , R_F , and R_R and assumed perfect operation of the masking, detection, isolation, and reconfiguration processes. As can be seen from this Table, there was no significant difference in system reliability for different values of R_M , R_F , and R_R . However, in the case summarized in Table 5.8, where C_F and C_R ranged from .9 to 1, each decrease in value of C_F and C_R resulted in a significant decrease in system reliability.

Finally, estimates computed from the fault tolerance performance model of the total time for fault detection and isolation and resumption of system processing with a spare replacement were inserted for the parameter δ in the phased mission model. Although these measured estimates resulted in a rate 50 times faster than the rate assumed in

Table 5.7. R_M , R_F , and R_R Results

λ_p	R_M	R_F	R_R	Probability of System Failure at 10 hours	Probability of System Failure at .5 hours
1×10^{-5}	6.5 ms	2.2 ms	18.7 ms	3.04×10^{-12}	2.59×10^{-15}
1×10^{-5}	1 sec	2.2 ms	18.7 ms	4.70×10^{-12}	8.5×10^{-14}
1×10^{-5}	6.5 ms	1 sec	18.7 ms	4.70×10^{-12}	8.58×10^{-14}
1×10^{-5}	6.5 ms	2.2 ms	1 sec	4.68×10^{-12}	8.44×10^{-14}
1×10^{-5}	1 sec	1 sec	1 sec	4.67×10^{-12}	8.37×10^{-14}

Table 5.8. C_F and C_R Results

λ_p	$\delta = R_F + R_R$	C_F, C_R	Probability of System Failure at 10 hours	Probability of System Failure at .5 hours
1×10^{-5}	21 ms	1	3.03×10^{-12}	2.14×10^{-15}
		.9999	2.99×10^{-8}	1.5×10^{-9}
		.999	2.99×10^{-7}	1.5×10^{-8}
		.99	2.99×10^{-6}	1.5×10^{-7}
		.9	2.99×10^{-5}	1.5×10^{-6}
1×10^{-5}	1 sec	1	4.67×10^{-12}	8.37×10^{-14}
		.9999	2.99×10^{-8}	1.5×10^{-9}
		.999	2.99×10^{-7}	1.5×10^{-8}
		.99	2.99×10^{-6}	1.5×10^{-7}
		.9	2.99×10^{-5}	1.5×10^{-6}

the initial phased mission analysis, the phase and total mission reliabilities were the same as before. This is in agreement with the results of the parameter sensitivity analyses and the detailed recovery state analysis.

For the cases analyzed in this case study, different values for R_M , R_F , and R_R had less impact on system reliability than different values of C_F and C_R . In other words, the effectiveness of the FDIR processes is of more importance than the performance for the given scenario. The next iteration in the modeling should therefore concentrate on conducting experiments with the fault tolerance performance model under various faulted conditions to enable accurate predictions to be made for C_F and C_R . In the initial model, the recovery time parameter, δ , did not include an R_M component. The results of the fault tolerance performance modeling underscored the fact that R_M is highly application-dependent. Since R_M is a significant contributor to fault latency, the fault tolerance performance model should be simulated for application programs with varying communications requirements to get good statistical data. Also, the transition from the masking state to the fault detection and isolation state is not conditioned by a probability of success since the assumption of Byzantine Resilience would result in the masking of any arbitrary fault. A future iteration in the modeling should include such a conditional probability to analyze the effect this additional component of coverage might have. This conditional probability would also be required to include in the model states where Byzantine Resilience has been lost. Such a loss would occur, for example, in the four network element configuration after the failure of a network element.

5.2.3.4. Conclusions

This case study demonstrated the usefulness of the fault tolerance performance model in the development and verification of reliability models. It also illustrated that certain reliability model parameter values that are usually not known until measurements can be made on prototype systems can be estimated from performance models. For the selected example, the design refinement modeling phase was useful in determining the parameters and assumptions that are critical to system reliability and which should be examined in closer detail throughout the design. The case study also highlighted the need to look more closely at interactions between application programs and the FDIR processes. This interaction is certain to play an increasingly decisive role in the effectiveness of proposed fault tolerance mechanisms as increased levels of parallelism are implemented. The overwhelming memory and computation requirements for the construction and solution of large, complex Markov models make it difficult to look at certain aspects of system behavior in sufficient detail to make accurate reliability predictions. It would therefore be desirable for statistically valid hierarchical methods of solution to be developed analogous to those for performance models.

6. Conclusions

During the two completed phases of the DAHPHRS program, selected portions of the Working Group Methodology framework for the development of dependable systems were instantiated and carried out. A large number of case studies were conducted to illustrate and explore specific issues of performance and reliability model construction, fidelity, and use at each of three design levels. These case studies are summarized in Tables 6.1 through 6.3 according to modeling and analysis areas illustrated and the particular architecture and algorithm used.

Table 6.1. Summary of Baseline Determination Case Studies

Modeling Area	Analysis Area	Algorithm	Architecture
Model Development	High-level Workload and Workload Distribution	WTA/TS	
		WAUCTION	
	Function Library	WTA/TS	
	Attribute Model	WTA/TS, WAUCTION	
			FIPP, Multimax Hypercube
		WAUCTION	
Performance Simulation	High-level Parallel Decomposition	Generic	Generic
		Matrix Multiply	Generic
		WTA/TS Based on Risk	
	Parametric Studies	WTA/TS	FIPP
	High-level Interprocessor Communications	WTA/TS	
Reliability Analysis	Spares		Generic
	Degraded Performance		Generic
	Network Reliability		Hypercube, FIPP, AOSP
Software Analytic Models	Risk Assessment	WTA/TS	

In these case studies, effort was focused on those facets likely to reveal weaknesses in the existing methods and tools or likely to yield payoffs in the form of refinements to the methodology for using the models. To this end, areas where the characteristics of complex systems are distinguished from more ordinary systems were considered. In Phase I, emphasis was placed on the impact of parallel processing issues on the methods and tools for performance analysis. In Phase II, the primary emphasis was on modeling fault-tolerant mechanisms and reliability analysis. Of particular interest in the performance analyses was the use of models to investigate the impact of network communications on performance, the impact of fault tolerance on communications, the speedup and effectiveness achievable through varying levels of parallelism as a function of both processor and communication speed and the existence of inherently sequential components of tasks, the effectiveness of various embeddings of algorithms

Table 6.2. Summary of Initial Design Case Studies

Modeling Area	Analysis Area	Algorithm	Architecture
Model Development	Attribute Model	WTA/TS	FTPP
	Functional Model	WTA/TS	Hypercube
		WTA/TS	Multimax
Performance Simulation	Engineering Model	WAUCTION	
	Interprocessor Communications	WTA/TS	FTPP
Functional Simulation	Parallel Decomposition	WTA/TS	FTPP
	Shared Memory	WTA/TS	Multimax
Measurement	Reconfiguration Algorithms		
	Model Development and Validation	WAUCTION	
	Mapping, Scheduling, Communications, and Resource Contention	WTA/TS	Hypercube
		WTA/TS	Multimax
Fault Tolerance Performance	Communications	WTA/TS	FTPP
	Reconfiguration Algorithms		
Reliability Analysis	Model Construction and Reduction		FTPP
	Sensitivity		FTPP
	Architecture Configuration		FTPP
	Phased Mission		FTPP

Table 6.3. Summary of Design Refinement Case Studies

Modeling Area	Analysis Area	Algorithm	Architecture
Model Development	Functional Model		FTPP
		FTPP FDI	
		FTPP Recon	
		FTPP Scheduler	
		FTPP Network Services	
		Fork and Join	
		RT Distributed OS Control and Interprocess Comm.	
Performance Simulation	Engineering Model		ROMMP Testbed
	Behavioral Model		FTPP System
	Average Network Utilization	Fork and Join	ROMMP
Functional Simulation	Client Utilization	Fork and Join	ROMMP
	Client Utilization	Fork and Join	ROMMP
Behavioral Simulation	OS Control and Interprocess Comm.	Fork and Join	ROMMP
Reliability Analysis	Fault Tolerance Performance and Effectiveness	Fork and Join	FTPP System
Reliability Analysis	Experimental Data		FTPP

in architectures, and the distribution of workload by function and by processing resource.

In addition, case studies were conducted to illustrate the use of models to support the analysis of load balancing, resource scheduling and contention, deadlock prevention, and the performance of fault tolerance mechanisms. More detailed models of intercomputer communications were explored for evaluating fault tolerance and task distribution mechanisms, and functional models were created to model shared memory, task-to-resource mappings, and scheduling mechanisms. Case studies also looked at the use of measurement to support model development and validation. It was found in Phase I that although high-level models were useful, the ability to quickly construct performance models at the right level of abstraction and for varying degrees of parallel decomposition was essential to being able to investigate multiple architecture and algorithm decomposition alternatives. It was found in Phase II that the use of reusable, generic models can reduce the amount of work involved and thus facilitate algorithm and architecture trade-off studies: Generic models of frequently used functions can be built, validated, and stored in a function library. The library can also contain rules for decomposing and transforming the models with respect to system characteristics such as communication patterns, parallel resources, or fault tolerance.

Results from Phase I models indicated that since speedup of a parallel over a sequential implementation is relative to the processing workload to communication workload ratio inherent in the function, methods of static analysis of algorithm and architecture granularity ratios should be considered. It was found in Phase II that the use of an attribute definition language and evaluator with hierarchical models allowed these ratios to be expressed and evaluated for different parallel decompositions, and that these static evaluations could be used to select alternatives to be assessed in greater detail via simulation.

During Phase I, a model of a weapon-to-target assignment algorithm based on an auction strategy was developed and workload predictions were made based on the model. These predictions were compared to the measured workload of an implementation of this algorithm. The predicted workload differed from the measured workload in ways that indicated the simple monotonic functions used to represent model parameters did not well characterize the actual workload. More complex functions for the models for this algorithm were developed in Phase II and resulted in improved workload predictions.

Also, a case study of modeling operating systems for multiprocessor systems was undertaken to demonstrate the need to include operating system effects in performance models of application algorithms. To examine modeling abstractions and fidelity issues for incorporating operating system characteristics into system performance

models, useful performance abstractions for operating systems were identified and demonstrated. This experiment also indicated the usefulness of a hierarchical approach and the need for model validation.

The reliability case studies based on SDI applications and a parallel, fault-tolerant architecture demonstrated a set of reliability analyses that can be used in developing a candidate design and establishing bounds on the performance and effectiveness that proposed fault-tolerant mechanisms have to attain to achieve particular levels of reliability. Three areas that present modeling problems were highlighted: parallelism, shared regions of connectivity, and communications. The creation and validation of models is complicated by the need to include recovery states and transitions that are difficult to define and to measure rates for. The solution of the models for long mission times is made more difficult by the occurrence of mixed statistical distributions of varying orders of magnitude and by the inability to use truncation techniques when a significant contribution to system failure is made by states reached through multiple failure occurrences. The requirement for a large number of analyses across a wide range of system parameters necessitates a model whose attributes can be expressed parametrically and instantiated for each analysis. It also requires quick computation of model solutions. The volume of output from a large number of analyses also requires some automated mechanism of organizing that output and selecting the pertinent information for each analysis. For the selected example, the modeling was useful in determining the parameters and assumptions that are critical to system reliability and which should be examined in closer detail throughout the design. The need to look more closely at interactions between application programs and the FDIR processes was highlighted. This interaction is certain to play an increasingly decisive role in the effectiveness of proposed fault tolerance mechanisms as increased levels of parallelism are implemented. The overwhelming memory and computation requirements for the creation and solution of large, complex Markov models make it desirable for statistically validated hierarchical methods of solution to be developed analogous to those for performance models.

Since the primary emphasis of Phase II was reliability analysis for highly parallel computer architectures and the modeling and evaluation of fault tolerance mechanisms that are inherent in highly reliable architectures, an integrated fault tolerance performance model of a parallel, fault-tolerant architecture was constructed and reliability analysis incorporating the results from the model was performed. The results of this effort demonstrate the potential value of behavioral modeling in support of fault tolerance evaluation prior to implementation. Fault injection experiments using this model were carried out and the results led to the identification of a fault containment problem and the determination of key FDIR parameters for use in reliability analysis. It also illustrated that certain reliability model parameter values that are usually not known until measurements can be made on prototype systems can be estimated from performance models.

In an effort to satisfy system requirements, a designer must be able to perform system trades that exercise a large number of design options. More accurate predictions of a fault-tolerant system's performance and reliability early in the design cycle facilitate the necessary trade-offs. In a parallel, fault-tolerant architecture, it is particularly necessary to be able to consider different workload-to-resource mappings to decide on parallel decompositions and recovery strategies, and several case studies in Phase I and II looked at how this could be accomplished.

With the completion of the Phase I and Phase II case studies, the need for models which combine effects and capture interactions of the architecture, operating system, application software, communication system services, and the hardware and software fault tolerance mechanisms is established and their value demonstrated.

Performance modeling is based on the analysis of multiple models of varying levels of detail and complexity. Therefore, parameterized, hierarchical models that allow different levels of abstraction to be represented and analyzed are required. The least detailed level of modeling requires tools that can perform a static analysis of performance characteristics based on specified attribute values of the models. At the next level, performance modeling tools must support the simulation of algorithmic processes mapped, or constrained, to a structural model of the architecture through the use of models that capture data and control flow. As more detailed analysis is required, the tools must be able to simulate functional models of either the algorithms or the architectural components. Finally, they must be able to incorporate various levels of system component models into a system behavioral model that 1) captures data flow, control flow, functionality, and changes in data value, 2) models the effects of operating system and scheduling functions, architectural components, fault tolerance mechanisms, and application algorithms, and 3) can represent fault effects.

The ability to construct experiments that can measure system processes to the degree required by the models constrains the level of detail to which system behavior can be modeled. Experimentation procedures and methods are necessary to accurately measure the most critical parameters. These methods include physical testing and simulation. One of the most important of the experimental methods is the injection of faults to observe system behavior and measure detection, isolation, and recovery times. Fault injection can be done with a working prototype of the system or with a gate-level model of hardware components. This work demonstrated that fault injection can also be done on a behavioral simulation model of the system, and that it can be done earlier in the design cycle while incorporating application and operating system effects.

Establishing reliability requires that a comprehensive set of qualitative and quantitative evaluations be conducted. On the quantitative side, realistic estimates of

important reliability model parameters such as fault detection coverage, the time required to recover from a fault after it has been detected, and failure rates must be established. Qualitative evaluations are required to assure that assumptions that are implicit in the construction of the reliability model are in fact satisfied. Examples of implicit assumptions include the assumption that the fault tolerance mechanisms are free of design faults and the assumptions regarding fault containment boundaries, types of faults, and synchronization of redundant channels.

The effort required to determine the reliability of a system to an acceptable level of confidence is driven to a large degree by the per-hour probability of failure. However, other factors contribute to the difficulty of establishing system reliability, including mission duration, system complexity, and characteristics of the failure processes. The level of confidence that can be placed in reliability predictions and measures varies with the accuracy of the data and the suitability of the techniques used to derive them, the validity of the modeling assumptions for the system being modeled, and the fault models considered. These factors are functions of the design-cycle stage at which the determination is made, of the overall level of reliability required of the system, and of the sensitivity of the system reliability to the performance and functionality of the fault tolerance mechanisms and assumed failure modes. As the fault tolerance of a system is increased to attain the ultra-high levels of reliability desired for mission- or life-critical systems, the task of determining system reliability becomes more difficult because it becomes more sensitive to the performance and functionality of the fault tolerance mechanisms and the validity of the assumed failure modes.

In the early- to mid-design phases, fault tolerance evaluation is of necessity incomplete due to lack of design detail. In the early phases, evaluation at best is restricted to qualitative assessments and parametric sensitivity analyses. However, it is crucial to begin these high-level analyses at the very first iteration to guide the design process. The focus is likely to be on the design implications of high-level fault tolerance requirements and identified fault classes for establishing bounds for relevant parameters and for distinguishing between broad classes of architectural design decisions. As the design progresses, fault tolerance evaluation can be more quantitative and the qualitative analyses can be more thorough. The focus can shift to finer discrimination between competing designs, establishing viability of a particular design, identifying design deficiencies and areas needing improvement, determining more realistic estimates of parameters such as recovery time or detection coverage, determining the behavior of fault tolerance mechanisms in the presence of faults, and finding and eliminating concept- or requirement-level design errors in the fault tolerance mechanisms.

The use of models throughout the design process is crucial to the development of efficient system architectures for applications requiring both high reliability and high throughput. Models provide a means of interaction between performance and reli-

bility analysis that allows the evaluation of a system's fault tolerance to be an integral part of system design from the earliest stages. In this effort, we have defined a three-phased modeling framework to support the iterative nature of the design process and to accommodate the changing information base and required measures and trade-offs as a design proceeds. Each phase assumes an iterative process of model construction, analysis, and refinement until the design space has been examined sufficiently to determine if a design can be expected to meet the system requirements. The measures that can be accomplished in a particular phase of modeling are dependent on the level of detail available for the components of the design. As the design progresses and more detailed information becomes available, the models developed at a previous phase can be refined and the evaluations focussed to produce the measures needed to proceed with the design.

We have also proposed a preliminary environment for the integration of tools based on common model generation, experiment management, and data base tools. This environment will enable the creation of the necessary models, ensure consistency between different models, facilitate the transfer of data among models, and enable the required analyses. With the completion of Phase II, requirements can be specified for the integration of existing tools with a common data base and experiment manager.

References

- [1] SDIO BM/C³ Processor and Algorithm Working Group. Application of Fault Tolerance Technology, Vols. I, II, III, & IV. October 1987. To be published by RADC (Task TA0006 of Contract F04701-85-C-0136).
- [2] C. O. Scheper, R. L. Baker, G. A. Frank, S. Yalamanchili, and F. G. Gray. Integration of Tools for the Design and Assessment of High-Performance, Highly Reliable Computing Systems (DAHPHRS). Interim Report. RADC-TR-90-81, RADC. May 1990.
- [3] John J. Shaw et al. *Battle Management Structures, Volume I: Directed Energy Weapon, Weapon Target Assignment Algorithms and Volume II: Kinetic Energy Weapon, Weapon Target Assignment Algorithms*. Technical Report, RADC-TR-89-84, RADC, January 1988.
- [4] Richard E. Harper. *Critical Issues in Ultra-Reliable Parallel Processing*. Technical Report CSDL-T-944, Charles Stark Draper Laboratory, Inc., Cambridge Massachusetts, June 1987.
- [5] Robert L. Baker. *Algorithm I Description*. Technical Report, Research Triangle Institute, Research Triangle Park, NC, May 1988.
- [6] Charlotte O. Scheper. *Algorithm II Description*. Technical Report, Research Triangle Institute, Research Triangle Park, NC, December 1988.
- [7] Gregory Jackson. *Teamwork Tutorial*. Cadre Technologies Inc., Providence, RI, July 1987.
- [8] Derek J. Hatley and Imtiaz A. Pirbhai. *Strategies for Real-Time System Specification*. Dorset House Publishing, New York, NY, 1987.
- [9] Ying W. Ng and A. Avizienis. A Unified Reliability Model for Fault-Tolerant Computers. *IEEE: Transactions on Computers*, C-29(11), November 1980.
- [10] S. L. Bavuso and P. L. Petersen. *CARE III Model Overview and User's Guide: First Edition*. NASA Technical Memorandum 86404, April 1985.
- [11] K. Trivedi, S. Bavuso, and et al. *HARP: The Hybrid Automated Reliability Predictor Introduction and Guide for Users*. NASA, September 1986.
- [12] Ricky W. Butler and Allan L. White. *SURE Reliability Analysis: Program and Mathematics*. NASA Technical Paper 2764, March 1988.
- [13] Sally C. Johnson. *ASSIST User's Manual*. NASA Technical Memorandum 87735, August 1986.

- [14] Ricky W. Butler and Philip H. Stevenson. *The PAWS and STEM Reliability Analysis Programs*. NASA Technical Memorandum 100572, March 1988.
- [15] C. J. Colbourn. *The Combinatorics of Network Reliability*. Oxford University Press, 1987.
- [16] Joanne Bechta Dugan, Malathi Veeraraghavan, Mark Boyd, and Nitin Mittal. Bounded Approximate Reliability Models for Fault Tolerant Distributed Systems. In *Proceedings of the 8th Symposium on Reliable Distributed Systems*, 1989.
- [17] Malathi Veeraraghavan. *Modeling and Evaluation of Fault-Tolerant Multiple Processor Systems*. PhD thesis, Department of Electrical Engineering, Duke University, August 1988.
- [18] Richard E. Harper. Reliability Analysis of Parallel Processing Systems. In *Proceedings of the 8th Digital Avionics Systems Conference*, pages 213–219, 1988.
- [19] Raytheon Company. *Advanced Onboard Signal Processor (AOSP) Phase IIA Development*. Technical Report RADC-TR-85-36, Vol. 1, Rome Air Development Center, February 1985.
- [20] J. A. Abraham. An Improved Algorithm for Network Reliability. *IEEE Transactions on Reliability*, R-28(1):58–61, April 1979.
- [21] K. K. Aggarwal, K. B. Misra, and J. S. Gupta. A Fast Algorithm for Reliability Evaluation. *IEEE Transactions on Reliability*, R-24:83–85, 1975.
- [22] R. G. Bennetts. On the Analysis of Fault Trees. *IEEE Transactions on Reliability*, R-24(3):194–203, August 1975.
- [23] W. H. Debany, P. K. Varshney, and C. R. P. Hartmann. Network Reliability Evaluation Using Probability Expressions. *IEEE Transactions on Reliability*, 161–165, June 1986.
- [24] S. Rai and K. K. Aggarwal. An Efficient Method for Reliability Evaluation of a General Network. *IEEE Transactions on Reliability*, 206–211, August 1978.
- [25] G. Fishman. A Comparison of Four Monte Carlo Methods for Estimating the Probability of s-t Connectedness. *IEEE Transactions on Reliability*, 145–155, June 1986.
- [26] S. Hariri and C. S. Raghavendra. SYREL: A Symbolic Reliability Algorithm Based on Path Cutset Methods. *IEEE Transactions on Computers*, C-36(10):1224–1232, October 1987.

- [27] A. Satyanarayana. A Unified Formula for the Analysis of Some Network Reliability Problems. *IEEE Transactions on Reliability*, R-31(1):23–32, 1982.
- [28] A. P. Wood. Multistate Block Diagrams and Fault Trees. *IEEE Transactions on Reliability*, 236–240, 1985.
- [29] Sally C. Johnson. em Evaluation of Fault-Tolerant Parallel-Processor Architectures over Long Space Missions. NASA Technical Memorandum 4123, 1989.
- [30] R. J. A. Buhr. *Systems Design with Ada*. Prentice-Hall, Inc., 1984.

DISTRIBUTION LIST

addresses	number of copies
RL/C3AA ATTN: Patrick O'Neill Griffiss AFB NY 13441-5700	10
Research Triangle Institute Center Digital Systems Research Research Triangle Park NC 27709	5
RL/DOVL Technical Library Griffiss AFB NY 13441-5700	1
Administrator Defense Technical Info Center DTIC-FDAC Cameron Station Building 5 Alexandria VA 22304-6145	2
Strategic Defense Initiative Office Office of the Secretary of Defense Wash DC 20301-7100	2
RL/C3AB Griffiss AFB NY 13441-5700	1
HQ USAF/SCTT Washington DC 20330-5190	1
SAF/AQSC Pentagon Rm 4D 269 Wash DC 20330	1

Naval Warfare Assessment Center
GIDEP Operations Center/Code 30G
ATTN: E Richards
Corona CA 91720

1

HQ AFSC/XTH
Andrews AFB MD 20334-5000

1

HQ SAC/SCPT
OFFUTT AFB NE 68046

2

HQ TAC/DRIY
ATTN: Maj. Divine
Langley AFB VA 23665-5575

1

HQ TAC/DOA
Langley AFB VA 23665-5554

1

ASD/ENEMS
Wright-Patterson AFB OH 45433-6503

1

SM-ALC/MACEA
ATTN: Danny McClure
Bldg 237, MASOF
McClellan AFB CA 95652

1

WRDC/AAAI-4
Wright-Patterson AFB OH 45433-6543

1

WRDC/AAAI-2
ATTN: Mr Franklin Hutson
WPAFB OH 45433-6543

1

1
AFIT/LD&E
Building 642, Area B
Wright-Patterson AFB OH 45433-6583

1
WRDC/MTEL
Wright-Patterson AFB OH 45433

1
AAMRL/HE
Wright-Patterson AFB OH 45433-6573

1
Air Force Human Resources Lab
Technical Documents Center
AFHRL/LRS-TDC
Wright-Patterson AFB OH 45433

1
AUL/LSE
Bldg 1405
Maxwell AFB AL 36112-5564

1
HQ AFSPACECOM/CNP
STINFO Officer
ATTN: Dr. W. R. Matoush
Peterson AFB CO 80914-5001

1
HQ ATC/TTOI
ATTN: Lt Col Killian
Randolph AFB TX 78150-5001

1
AFLMC/LGY
ATTN: Maj. Shaffer
Building 205
Gunter AFS AL 36114-6693

1
US Army Strategic Def
CSSD-IM-PA
PO Box 1500
Huntsville AL 35807-3801

Ofc of the Chief of Naval Operation
ATTN: William J. Cook
Navy Electromagnetic Spectrum Mgt
Room 5A678, Pentagon (OP-941)
Wash DC 20350

1

Commanding Officer
Naval Avionics Center
Library D/765
Indianapolis IN 46219-2139

1

Commanding Officer
Naval Ocean Systems Center
Technical Library
Code 96423
San Diego CA 92152-5000

1

Cmdr
Naval Weapons Center
Technical Library/C3431
China Lake CA 93555-6001

1

Superintendent
Code 524
Naval Postgraduate School
Monterey CA 93943-5000

1

Space & Naval Warfare Systems Comm
Washington DC 20363-5100

1

CDR, U.S. Army Missile Command
Redstone Scientific Info Center
14SMI-RD-CS-0/ILL Documents
Redstone Arsenal AL 35898-5241

2

Advisory Group on Electron Devices
Attn: Documents
2011 Crystal Drive, Suite 307
Arlington VA 22202

2

Los Alamos National Laboratory
Report Library
MS 5000
Los Alamos NM 87544

1

AEDC Library
Tech Files/MS-100
Arnold AFB TN 37389

1

Commander, USAG
ASOH-PCA-CRL/Tech Lib
Bldg 61801
Ft Huachuca AZ 85613-6000

1

1839 EIG/EIT
Keesler AFB MS 39534-6348

1

AFEWC/ESRI
San Antonio TX 78243-5000

3

ESD/XRR
Hanscom AFB MA 01731-5000

1

SEI JPO
ATTN: Major Charles J. Ryan
Carnegie Mellon University
Pittsburgh PA 15213-3890

1

Director NSA/CSS
T5122/TDL
ATTN: D W Marjarum
Fort Meade MD 20755-6000

1

Director NSA/CSS
W157
9800 Savage Road
Fort Meade MD 20755-6000

1

NSA
ATTN: D. Alley
Div X911
9800 Savage Road
Ft Meade MD 20755-6000

1

Director
NSA/CSS
W11 DEFSMAC
ATTN: Mr. Mark E. Clesh
Fort George G. Meade MD 20755-6000

1

Director
NSA/CSS R12
ATTN: Mr. Dennis Heinbuch
9800 Savage Road
Fort George G. Meade MD 20755-6000

1

DoD
R31
9800 Savage Road
Ft. Meade MD 20755-6000

1

DIRNSA
#509
9300 Savage Road
Ft Meade MD 20775

1

Director
NSA/CSS
R08/R & E BLDG
Fort George G. Meade MD 20755-6000

1

DOD Computer Center
C/TIC
9300 Savage Road
Fort George G. Meade MD 20755-6000

1

ESD/AV
HANSCOM AFB MA 01731-5000

1

ESD/IC
HANSCOM AFB MA 01731-5000

1

FL 2307/RESEARCH LIBRARY
OL AA/SULL
HANSCOM AFB MA 01731-5000

1

Technical Reports Center
Mail Drop D130
Burlington Road
Bedford MA 01731

1

Defense Technology Sec Admin (DTSA)
ATTN: STTD/Patrick Sullivan
400 Army Navy Drive
Suite 300
Arlington VA 22202

1

SDI/S-PI-BM
ATTN: Cmdr Korajo
The Pentagon
Wash DC 2031L-7100

1

SDIO/S-PI-9M
ATTN: Capt Johnson
The Pentagon
Wash DC 20301-7000

1

SDIO/S-PI-BM
ATTN: Lt Col Rindt
The Pentagon
Wash DC 20301-7100

1

SDI Technical Information Center
1755 Jefferson Davis Highway #708
Arlington VA 22202

1

SAF/AQSD
ATTN: Maj M. K. Jones
The Pentagon
Wash DC 20330

1

AFSC/CV-D
ATTN: Lt Col Flynn
Andrews AFB MD 20334-5000

1

HQ SD/XR
ATTN: Col Heimach
PO Box 92960
Worldway Postal Center
Los Angeles CA 90009-2960

1

HQ SSD/CNC
ATTN: Col O'Brien
PO Box 92960
Worldway Postal Center
Los Angeles CA 90009-2960

HQ SD/CNCI
ATTN: Col Collins
PO Box 92960
Worldway Postal Center
Los Angeles CA 90009-2960

ESD/AT
ATTN: Col Ryan
Hanscom AFB MA 01731-5000

ESD/ATS
ATTN: Lt Col Oldenberg
Hanscom AFB MA 01731-5000

ESD/ATN
ATTN: Col Leib
Hanscom AFB MA 01731-5000

AFSTC/XPX (Lt Col Detucci)
Kirtland AFB NM 87117

AFSPACECOM/XPD
ATTN: Maj Roger Hunter
Peterson AFB CO 80914

SSD/CNI
ATTN: Lt Col Joe Rouge
P. O. Box 92960
Los Angeles AFB CA 90009-2960

NTS JPO
ATTN: Maj Don Ravenscroft
Falcon AFB CO 80912

Naval Air Development Ctr
ATTN: Dr. Mort Metersky
Code 3CD
Warminster PA 139974

1

HQ AFOTEC/OAHS
ATTN: Dr. Samuel Charlton
Kirtland AFB NM 87117

1

ESD/XTS
ATTN: Lt Col Joseph Toole
Hanscom AFB MA 01731

1

SDIO/ENA
ATTN: Col R. Worrell
Pentagon
Wash DC 20301

1

USA-SDC CSSD-H-SBE
ATTN: Mr. Doyle Thomas
Huntsville AL 35807

1

HQ AFSPACECOM/DOXP
ATTN: Capt Mark Terrace
Stop 7
Peterson AFB CO 80914

1

ESD/XTI
ATTN: Lt Col Paul Monico
Hanscom AFB MA 01730

1

CSSD-H-SS
ATTN: Mr. Larry Tubbs
Commander USA SDC
PO Box 1500
Huntsville AL 35807

1

USSPACECOM/J5B
ATTN: Lt Col Harold Stanley
Peterson AFB CO 80914

1

4T9 JPO 1
ATTN: Mr. Nat Sojourner

Falcon AFB CO 80912

RADC/C3A 1
ATTN: Mr. Anthony F. Snyder

Griffiss AFB NY 13441

NASA Langley Research Center 1
ATTN: Wayne Bryant

MS578

Hampton VA 23665-5225

AF Space Command/XPIXIS 1
Peterson AFB CO 80914-5001

AFOTEC/XPP 1
ATTN: Capt Wrobel

Kirtland AFB NM 87117

Director NSA (V31) 1
ATTN: George Hoover

9300 Savage Road

Ft George G. Meade MD 20755-6000

SSD/CNIR 1
ATTN: Capt Brandenburg

PO BOX 92960-2960

LOS ANGELES CA 90009-2960

NASA Langley Research Center 2
ATTN: Sally Johnson

MS130

Hampton VA 23665-5225

Trusted Information Systems, Inc. 1
ATTN: Richard E. Schwinger

P.O. Box 45

3060 Washington Rd

Glenwood MD 21738

SRI International 1
ATTN: Darlene Sherwood
For: Comp Sci Lao/John Rushby
333 Ravenswood Ave
Menlo Park CA 94025

Institute for Defense Analysis 1
ATTN: Dr. Russell Fries
Comp & Soft Eng Div/W. Mayfield
1801 N. Beauregard Street
Alexandria VA 22311-1772

Secure Computing Technology Corp 1
ATTN: Jerry A. Herby
1210 W. County Rd Es Suite 100
Arden Hills MN 55112

SRI International 1
ATTN: Darlene Sherwood
For: Comp Sci Lab/Teresa Lunt
333 Ravenswood Ave
Menlo Park CA 94025

The Aerospace Corp/Def Develop Div 1
ATTN: James G. Gee
For: George Gilley, ML-046
P.O. Box 92957
Los Angeles CA 90009-2957

GE Co/Strategic Systems Dept 1
ATTN: Tim Pawlik
For: Bill Bensch
1787 Sentry Park W., PO Box 1000
Bluebell PA 19422

GE Co/Strategic Systems Dept 1
ATTN: Tim Pawlik
For: Mr. Ron Marking
1787 Sentry Park W., PO Box 1000
Bluebell PA 19422

Advisory Group on Electron Devices 1
201 Varick Street, Rm 1142
New York NY 10014

MITRE Corp 1
ATTN: Dr. Donna Cuomo
Bedford MA 01730

Ford Aerospace Corp
c/o Rockwell International
ATTN: Dr. Joan Schulz
1250 Academy Park Loop
Colorado Springs CO 80910

1

Essex Corp
ATTN: Dr. Bob Mackie
Human Factors Research Div
5775 Dawson Ave
Goleta CA 93117

1

RJO Enterprises
ATTN: Mr. Dave Israel
1225 Jefferson Davis HY
Suite 300
Arlington VA 22202

1

BBN Systems & Technology
ATTN: Dr. Dick Pew
70 Fawcett St
Cambridge MA 02138

1

Bonnie McDaniel, MDE
313 Franklin St
Huntsville AL 35801

1

Harris Corp
Government Info Sys Division
ATTN: Ronda Henning
PO Box 98000
Melbourne FL 32902

1

Ford Aerospace & Comm Corp
ATTN: Peter Baze (Mail Stop 29A)
10440 State Highway 83
Colorado Springs CO 80908

1

Computational Logic, Inc.
ATTN: Dr. Donald I. Good
1717 W. 6th St (Suite 29C)
Austin TX 78703

1

Gemini Computers Inc.
ATTN: Roger Schell
2511 Garden Rd (Bldg C, Ste 1000)
Monteray CA 93940

1

Boeing Aerospace Co
ATTN: Dan Schnackenberg (MS 38-12)
P.O. Box 3999
Seattle WA 98124-2499

1

BBN Laboratories, Inc.
ATTN: Steve Vinter
10 Moulton Street
Cambridge MA 02238

1

Univ of Calif at Santa Barbara
Computer Science Dept
ATTN: Prof Richard A. Kemmerer
Santa Barbara CA 93106

1

Unisys Corp
ATTN: Deborah Cooper
5731 Slauson Ave
Culver City CA 90230

1

Philips Lab/STET
ATTN: Roe Maier
Kirkland AFB NM 87117-6008

1

**MISSION
OF
ROME LABORATORY**

Rome Laboratory plans and executes an interdisciplinary program in research, development, test, and technology transition in support of Air Force Command, Control, Communications and Intelligence (C³I) activities for all Air Force platforms. It also executes selected acquisition programs in several areas of expertise. Technical and engineering support within areas of competence is provided to ESD Program Offices (POs) and other ESD elements to perform effective acquisition of C³I systems. In addition, Rome Laboratory's technology supports other AFSC Product Divisions, the Air Force user community, and other DOD and non-DOD agencies. Rome Laboratory maintains technical competence and research programs in areas including, but not limited to, communications, command and control, battle management, intelligence information processing, computational sciences and software producibility, wide area surveillance/sensors, signal processing, solid state sciences, photonics, electromagnetic technology, superconductivity, and electronic reliability/maintainability and testability.